HIGH QUALITY COMPACT DELAY TEST GENERATION

A Dissertation

by

ZHENG WANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2010

Major Subject: Computer Engineering

HIGH QUALITY COMPACT DELAY TEST GENERATION

A Dissertation

by

ZHENG WANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Duncan M. Walker
Committee Members,	Jianer Chen
	Rabinarayan Mahapatra
	Weiping Shi
Head of Department,	Valerie E. Taylor

May 2010

Major Subject: Computer Engineering

ABSTRACT

High Quality Compact Delay Test Generation.

(May 2010)

Zheng Wang, B.S., Zhejiang University, China;

M.S., Zhejiang University, China

Chair of Advisory Committee: Dr. Duncan M. Walker

Delay testing is used to detect timing defects and ensure that a circuit meets its timing specifications. The growing need for delay testing is a result of the advances in deep submicron (DSM) semiconductor technology and the increase in clock frequency. Small delay defects that previously were benign now produce delay faults, due to reduced timing margins. This research focuses on the development of new test methods for small delay defects, within the limits of affordable test generation cost and pattern count.

First, a new dynamic compaction algorithm has been proposed to generate compacted test sets for K longest paths per gate (KLPG) in combinational circuits or scan-based sequential circuits. This algorithm uses a greedy approach to compact paths with non-conflicting necessary assignments together during test generation. Second, to make this dynamic compaction approach practical for industrial use, a recursive learning algorithm has been implemented to identify more necessary assignments for each path, so that the path-to-test-pattern matching using necessary assignments is more accurate.

Third, a realistic low cost fault coverage metric targeting both global and local delay faults has been developed. The metric suggests the test strategy of generating a different number of longest paths for each line in the circuit while maintaining high fault coverage. The number of paths and type of test depends on the timing slack of the paths under this metric. Experimental results for ISCAS89 benchmark circuits and three industry circuits show that the pattern count of KLPG can be significantly reduced using the proposed methods. The pattern count is comparable to that of transition fault test, while achieving higher test quality. Finally, the proposed ATPG methodology has been applied to an industrial quad-core microprocessor. FMAX testing has been done on many devices and silicon data has shown the benefit of KLPG test.

DEDICATION

To my parents:

without their support this would not have been possible

ACKNOWLEDGEMENTS

I would like to thank my advisor and committee chair, Dr. Duncan M. (Hank) Walker for his advice and support throughout my Ph.D. studies at Texas A&M University. His insights in this particular research area, his technical guidance and spiritual support were invaluable to this work. This dissertation would never have been completed without his advice and encouragement. I owe him lots of gratitude for making my research life enjoyable and rewarding. What I learned from him will benefit my future career.

I am also grateful to my committee members, Dr. Jianer Chen, Dr. Rabinarayan Mahapatra and Dr. Weiping Shi, for their valuable suggestions and personal encouragement.

My gratitude also goes to my lab members: Wangqi Qiu, Zhongwei Jiang, Sivakumar Ganesan, Lei Wu, Jing Wang and Shayak Lahiri, for their friendship and help. Especially I want to thank Wangqi Qiu for his help and advice in the beginning of my research, and Zhongwei Jiang and Sivakumar Ganesan for their great contribution to the industrial experiments.

Also, I want to acknowledge the Semiconductor Research Corporation (SRC) and National Science Foundation (NSF) for their financial support of this research.

Finally, I would like to thank my family for giving me company during my study. Their love enriched my life. Without their constant support, this would not have been a wonderful journey.

TABLE OF CONTENTS

Page

ABSTRACT iii

DEDICATION v

ACKNOWLEDGEMENTS vi

TABLE OF CONTENTS vii

LIST OF FIGURES ix

LIST OF TABLES xi

1. INTRODUCTION 1

    1.1 Delay Testing 1
    1.2 Delay Fault Models 1
    1.3 Scan Based Test 4
    1.4 KLPG Test Generation 9
    1.5 Structure of The Dissertation 13

2. DYNAMIC COMPACTION FOR COMPACT DELAY TEST GENERATION 14

    2.1 Motivation 14
    2.2 Previous Work 15
    2.3 Proposed Dynamic Compaction Algorithm 19
    2.4 Dynamic Compaction for KLPG Test 25
    2.5 Experimental Results for KLPG Test 26
    2.6 Experimental Results for Transition Fault Test and and Stuck-at Test .39
    2.7 Conclusions 44

3. IMPROVED DYNAMIC COMPACTION WITH RECURSIVE LEARNING 45

    3.1 Motivation 45
    3.2 Previous Work 47

Page

3.3 Improved Dynamic Compaction with Recursive Learning 51
3.4 Experimental Results 53
3.5 Conclusions 55

4. DELAY TEST GENERATION WITH A REALISTIC LOW COST FAULT COVERAGE METRIC 56

4.1 Motivation 56
4.2 Previous Work 58
4.3 Low Cost Fault Coverage Metric 60
4.4 Realistic Low Cost Fault Coverage Metric 62
4.5 KLPG with Realistic Low Cost Fault Coverage Metric 66
4.6 Experimental Results 70
4.7 Conclusions 77

5. EXPERIMENTS ON SILICON 78

5.1 Flow of AMD Experiments 78
5.2 Improved KLPG for AMD Design 80
5.3 Experimental Results 83

6. SUMMARY AND FUTURE WORK 92

6.1 Summary 92
6.2 Future Work 93

REFERENCES 96

VITA 111

LIST OF FIGURES

Page

Figure 1 Structure of a scan design 5

Figure 2 Muxed-D scan cell 6

Figure 3 Shift register latch 7

Figure 4 Launch-on-shift clock waveform 9

Figure 5 Launch-on-capture clock waveform 9

Figure 6 Fault types addressed in current research 10

Figure 7 KLPG path generation algorithm 12

Figure 8 Static compaction of two 8-bit vectors 16

Figure 9 Greedy static compaction flow 17

Figure 10 Vector pair and necessary assignments (circles) for Path1 20

Figure 11 Vector pair and necessary assignments (Xs) for Path2 21

Figure 12 Vector pair and necessary assignments for Path1 and Path2 21

Figure 13 Pseudo code of dynamic compaction algorithm 23

Figure 14 Flowchart of dynamic compaction algorithm 24

Figure 15 Test generation flow with dynamic compaction 25

Figure 16 Robust sensitization criterion for OR gate 35

Figure 17 Non-robust sensitization criterion for OR gate 36

Figure 18 Functional sensitization criterion for OR gate 37

Figure 19 Test composition of KLPG-1 38

Figure 20 Transition fault test generation algorithm 41

Page

Figure 21 Direct implication examples 47

Figure 22 Example of recursive learning 49

Figure 23 Recursive learning algorithm 50

Figure 24 Improved dynamic compaction algorithm 52

Figure 25 Path statistics for ISCAS89 circuits 57

Figure 26 Path delay distribution of s38417 57

Figure 27 Fault coverage distribution 61

Figure 28 Delay space under different path correlations 62

Figure 29 Fault site with short and long paths 63

Figure 30 Fault site with long paths 63

Figure 31 Fault site with short paths only 64

Figure 32 Example of delay vs. bridge resistance 65

Figure 33 KLPG flow with low cost coverage metric 67

Figure 34 Fault coverage vs. K (circuit c7552) 70

Figure 35 KLPG flow for AMD design 80

Figure 36 Example of clock path justification 82

Figure 37 FMAX distribution for robust KLPG test (Core0) 86

Figure 38 FMAX distribution for robust KLPG test (Core1) 87

Figure 39 FMAX distribution for robust KLPG test (Core2) 88

Figure 40 FMAX distribution for robust KLPG test (Core3) 89

Figure 41 FMAX distribution for robust KLPG test (chip level) 90

LIST OF TABLES

		Page
Table 1	Circuits used in experiments	27
Table 2	Comparison of KLPG (K=1) test size with static and dynamic compaction (robust test LOC).	29
Table 3	Comparison of KLPG (K=1) test size with static and dynamic compaction (robust test LOS).	30
Table 4	Comparison of KLPG (K=1) test size with static and dynamic compaction (non-robust test LOC)	31
Table 5	Comparison of KLPG (K=1) test size with static and dynamic compaction (non-robust test LOS).	32
Table 6	POOL size influence on compaction (K=1 robust test)	34
Table 7	KLPG-1 vs. commercial tool	39
Table 8	TF vs. KLPG-1	43
Table 9	Final justification failure rate	46
Table 10	Improved dynamic compaction with recursive learning (LOC robust test, POOL = 1000)	54
Table 11	KLPG with low cost fault coverage metric (LOC with 20% process variation)	72
Table 12	KLPG with low cost fault coverage metric (LOC with 30% process variation)	73
Table 13	KLPG-5 vs. KLPG-5L	76
Table 14	KLPG-1 test for module 1 & module 2	83
Table 15	Robust KLPG results for AMD microprocessor	85

1. INTRODUCTION

1.1 Delay Testing

Timing is crucial with the increasing speed of integrated circuits and the advances in semiconductor fabrication technology. Most defects affecting the performance are gross functional defects that can be detected using traditional test methods [1][2][3]. However, some smaller manufacturing defects do not cause functional failure but only influence the circuit speed. A typical example is the spot defect that causes a resistive open or short. The International Technology Roadmap for Semiconductors (ITRS) [4] projects that small delay defects (SDD) that previously were benign now produce delay faults, due to reduced timing margins. Delay testing is used to detect those timing defects to ensure that a circuit meets its timing specifications. This is essential to achieve acceptable product quality. The delay test challenge is more difficult for chips fabricated in deep submicron (DSM) semiconductor technology with increased delay variability, signal crosstalk, power supply noise and temperature variations.

1.2 Delay Fault Models

A *defect* in a circuit is the unintended difference between the actual circuit implementation and the specification. A *fault* is the representation of a defect at the

This dissertation follows the style and format of *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

abstracted function level. A *fault model* is an abstraction of a type of defect behavior. Some popular delay fault models are discussed in the following sections.

1.2.1 Transition Fault Model

The transition fault (TF) model [5] is the most commonly used delay fault model. It assumes that the delay fault affects only one place in the circuit. In this model, each gate is assumed to have two transition faults: a slow-to-rise (STR) and a slow-to-fall (STR) delay fault. Thus the fault space of transition fault test is linear in the number of gates in circuit. The extra delay introduced by the transition fault is assumed to be large enough to prevent the transition from reaching any observable primary outputs within the specified time. In other words, the transition fault effect can be observed through any path (whether long or short) to any observable primary output.

Stuck-at fault test generation tools can be easily extended to generate tests for transition faults [2]**.** A transition fault test vector pair *{v1,v2}* can be composed by pairing stuck-at-0 and stuck-at-1 test patterns. The first vector *v1* initializes the circuit and the second vector *v2* sensitizes and propagates the fault effect to some observable primary outputs. Any stuck-at fault is covered by a corresponding transition fault test, since a stuck-at fault can be considered a very slow transition fault.

The main disadvantage of the TF model is that the size of the fault is not considered. Transition fault test generators normally select the easiest path, which is the shortest one in most cases, to activate and propagate a transition. Thus the quality of TF test for small delay defects is a concern [6][7]. Another problem is that TF test often propagates a glitch from the fault site [8], which introduces potential quality loss.

1.2.2 Gate Delay Fault Model

The gate delay fault model [9][10][11] [12] assumes that a spot defect is lumped on a gate input or output and takes into account the size of the extra delay. Detecting such faults requires testing a long path through the fault site. It is necessary to specify the delay fault size in order to determine the quality of a test set, which is defined by how close the minimum detected delay fault sizes are to the minimum detectable fault sizes.

1.2.3 Line Delay Fault Model

The line delay fault model [13][14] is a variation of the gate delay fault model. It requires testing a rising or falling delay fault through the longest sensitizable path on every line in the circuit. Sensitizing the longest path through the target line can detect the smallest delay defect on the target line. However, this model may fail to detect some defects [15] with the increase of process variation in new technologies [16].

1.2.4 Path Delay Fault Model

The path delay fault model [17] models the distributed delay on a path. It is the most conservative model since the fault space is all paths in the circuit. This model assumes that any path can have any delay. A circuit is considered faulty with a path delay fault if any one path is slow for a rising or falling transition. Thus tests for the path delay fault model can catch small distributed delay defects in the circuit. The primary limitation of the path delay fault model is that the number of paths in the circuit can be exponential in the number of gates. For this reason it is not practical to test all paths in

the circuit and achieve high test coverage. For example, ISCAS85 benchmark circuit c6288, a 16-bit multiplier has close to 10^{20} paths [18].

1.3 Scan Based Test

Design-for-Test (DFT) circuitry is inserted to enhance the testability of a circuit. Scan design is the most widely used DFT technique. Selected storage elements, such as latches and flip-flops, are connected together into scan chains to provide direct access. All selected storage elements are replaced with scan cells, each having one additional scan input (SI) port and one additional scan output (SO) port. All scan cells are formed by connecting the SO port of one scan cell to the SI port of the next scan cell. All scan cells can be set to a desired state by shifting specific values into scan chains. Similarly, the state of all scan cells can be observed by shifting the contents out of the scan chains. Therefore, the controllability and observability of the circuit is enhanced. Figure 1 shows the basic structure of a scan design. The circuit consists of combinational logic and a scan chain. In the test mode, the test data is applied to the circuit under test (CUT) through primary inputs (PI) *x1*, *x2* and scan cell outputs *a*, *b*, and *c*. The circuit responses are captured through primary outputs (PO) *z1*, *z2* and scan cell inputs *a'*, *b'*, and *c'*.

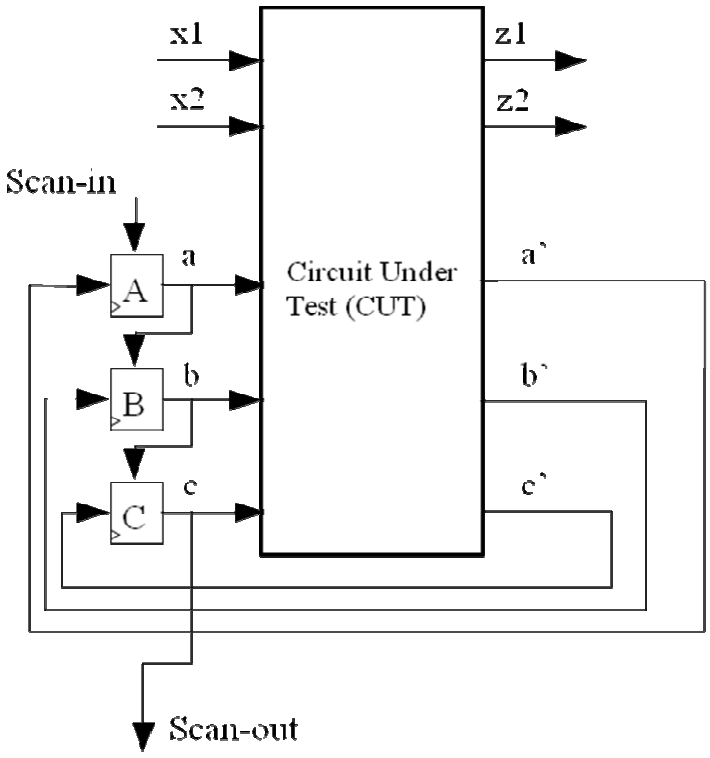


Figure 1. Structure of a scan design.

1.3.1 Scan Cell Type

There are several fundamental scan architectures: muxed-D scan, LSSD scan and enhanced scan.

A. Muxed-D Scan

Figure 2 shows an edge-triggered muxed-D scan cell design. The scan cell is composed of a multiplexer and a standard D flip-flop. The scan enable (SE) signal controls the multiplexer to select between the data input (D) and scan input (SI). Clock signal (CP) is used to clock the flip-flop in both normal and test modes.

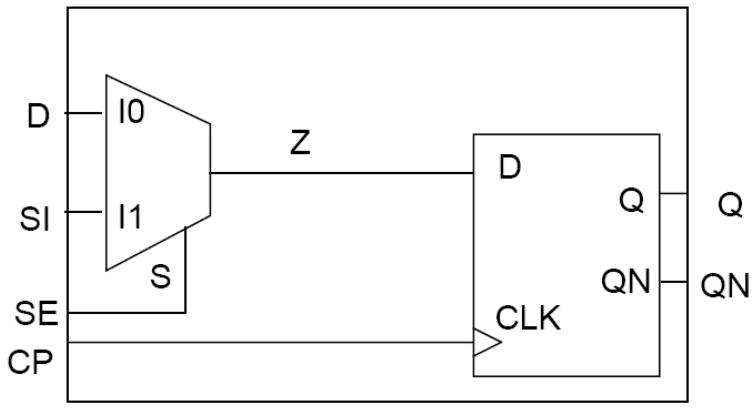


Figure 2. Muxed-D scan cell.

B. LSSD Scan

A shift register latch (SRL) [19][20] can be used as a level sensitive scan design (LSSD) scan cell. This scan cell contains a pair of latches, a master two-port D latch L_1 and a slave D latch L_2. Clocks *C*, *A* and *B* are used to select between the data input *D* and the scan input *I* to drive $+L_1$ and $+L_2$, as shown in Figure 3. During test the SRLs are accessed by applying appropriate clock signal sequences. LSSD can be implemented using a single-latch design [19] or a double-latch design [21] based on different clock schemes.

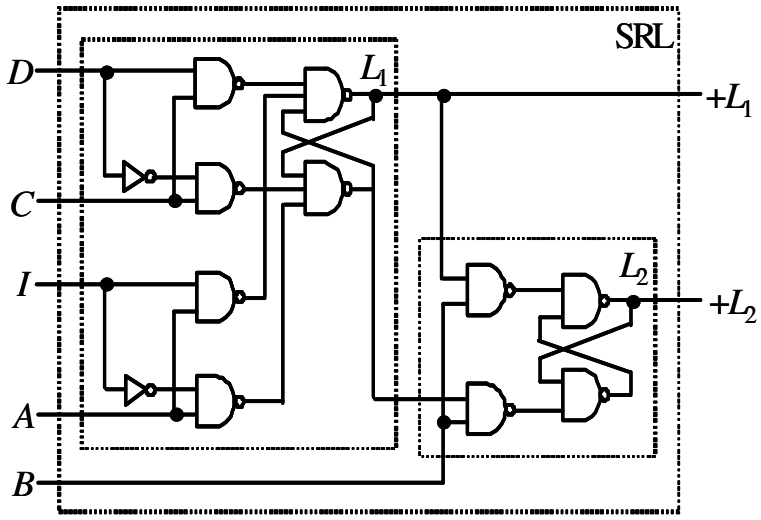


Figure 3. Shift register latch.

C. Enhanced Scan

Enhanced scan [22][23] allows storing two bits of data in the scan cell. Thus both initialization and test vector can be loaded into a scan cell and applied consecutively to the circuit under test. For a filp-flop design, this is achieved by adding an extra holding latch to the output of each flip-flop. Since the two bits are independent of one another, high fault coverage can be achieved by applying any arbitrary pair of test vectors. The main disadvantage of enhanced scan design is the extra area, timing and power introduced by the extra holding latch.

1.3.2 At-Speed Scan Clocking

There are two basic scan clocking schemes widely used in the industry for testing transition and path delay faults at-speed: launch-on-shift (or skewed-load [24][25]) and launch-on-capture (or broad-side test [26]).

In launch-on-shift (LOS) mode, the initialization vector is first scanned into the scan chains using the shift clock. The last shift clock launches the transition to the circuit and a following fast system clock captures the circuit response, as shown in Figure 4. For a test vector pair *{v1,v2}*, *v2* is derived from shifting *v1* by one bit. The primary advantages of LOS are that the test is derived in one time frame, simplifying test generation and reducing test generation time, and the pattern count is relatively low. The primary disadvantage of LOS is that the scan enable (SE) signal must be switched between the shift and system clock pulses at the rated system clock speed to capture the test result on the next system clock cycle. This requires that the scan enable be distributed via a fast clock tree. A second disadvantage of LOS is that it tests more delay faults that do not cause timing failure, and it cannot test some faults that do cause timing failure.

In launch-on-capture (LOC) mode, two consecutive system clock pulses are used to launch the transition and capture the circuit response, as shown in Figure 5. Dummy clock cycles are inserted as needed to give the SE time to switch from scan to capture mode. For a test vector pair *{v1,v2}*, *v2* is derived from the circuit response of the initialization vector *v1*. The advantages of the LOC approach are that there is no timing constraint on SE, and the test vector is a legal state of the circuit, assuming the initialization vector is a legal state. The disadvantages are that test generation is over two time frames, so takes more CPU time, and the test pattern count is higher than LOS test.

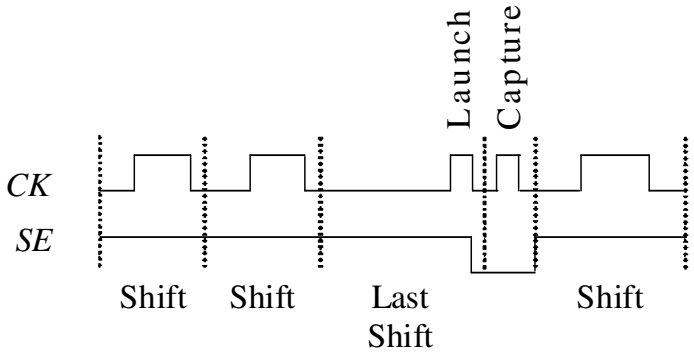


Figure 4. Launch-on-shift clock waveform.

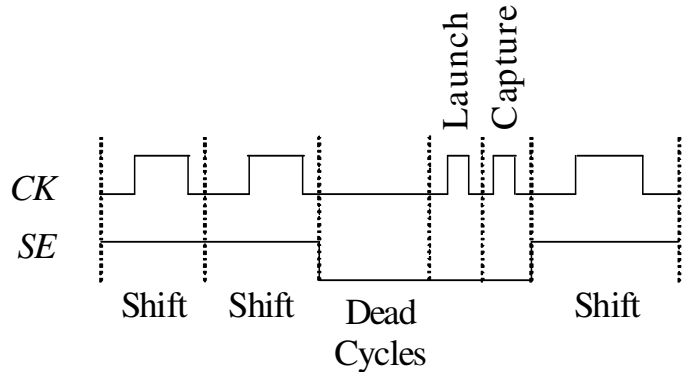


Figure 5. Launch-on-capture clock waveform.

1.4 KLPG Test Generation

Figure 6 shows the categorization of faults targeted in this research. *Local delay faults* are increases in circuit delay caused by a spot defect such as a resistive bridge or open. *Global delay faults* are slow paths due to die-to-die process parameter variation, such as metal thickness variation [27]. *Combined delay faults* are caused by a combination of spot defect and process variation. Process variation consists of systematic, die-to-die random variation and intra-die random variation. Systematic

process variation due to subwavelength lithography is assumed to be incorporated into circuit delay models, and not considered explicitly in this research.

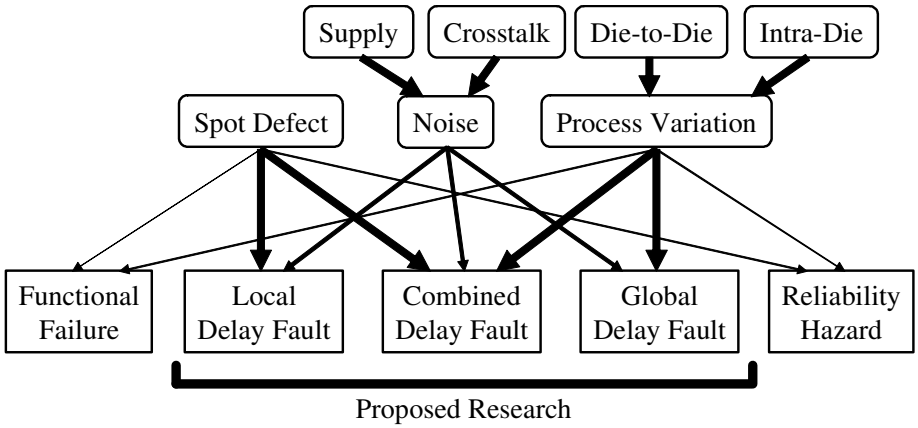


Figure 6. Fault types addressed in current research.

In [28], an efficient automatic test pattern generation (ATPG) algorithm was developed to generate the $2 \cdot K$ longest paths through each line in the combinational circuit, with K paths having a rising transition and K paths having a falling transition at the fault site. This work was later extended to sequential circuits in [29]. Figure 7 shows the basic flow of the algorithm.

We define a *launch point* as a primary (PI) or pseudo-primary input (PPI), and a *capture point* as a primary output (PO) or pseudo-primary output (PPO). In the preprocessing phase, topology information such as the *PERT delay* of each gate is calculated, which will help accelerate the path generation. The min-max *PERT delay* of a

gate is the min-max delay from this gate to capture points, without considering any logic constraints. Delays are extracted from a standard delay format (SDF) delay file.

In the path generation phase, a *path store* is used to store *partial paths*, which are paths originating from a launch point but have not reached a capture point. Because partial paths are initialized from launch points, each partial path initially contains only a PI or PPI. Every partial path has a value called *esperance* [30], which is the sum of the length of the partial path and the *PERT delay* from its last node to a capture point. In other words, the max esperance of a partial path is the upper bound of its delay when it reaches a capture point and becomes a complete path, and the min esperance is the lower bound. As shown in Figure 7, in each iteration of path generation, the partial path with the largest max esperance is popped from the sorted path store and extended by adding one fanout gate with largest max esperance. If the last gate of the partial path has multiple fanouts, the path will split, leaving the alternate choices in the path store. Depending on the sensitization criterion, such as *robust* or *non-robust* sensitization [31], constraints to propagate the transition on the added gate are applied. Then *direct implications* [28] are performed to identify local conflicts. A direct implication on a gate is one where the input or output value of that gate can be determined from other input or output values assigned to that gate. Previous research [28][30] found that direct implications can eliminate most false paths. Heuristics such as *forward trimming* and *smart-PERT delay* [28] are applied to the partial path in order to quickly eliminate false paths. These heuristics enable for the first time generation of the longest paths through every gate of ISCAS85 benchmark circuit c6288. All prior path delay test approaches

have failed on this circuit, due to its exponential number of long false paths. If a partial path reaches a capture point, it becomes a complete path. Then a PODEM-based *final justification* [28][29][32] is performed to find a vector pair that sensitizes this path. Since the longest path through one line may be the longest path through other lines, a new complete path must be checked to see if it has already been generated before. In order to utilize the overlapping paths between different lines to accelerate test generation, global longest path generation [28] is performed at the beginning of test generation. The test generation repeats until the K longest testable paths (both rising and falling transitions) through each line are generated or the path store is exhausted.

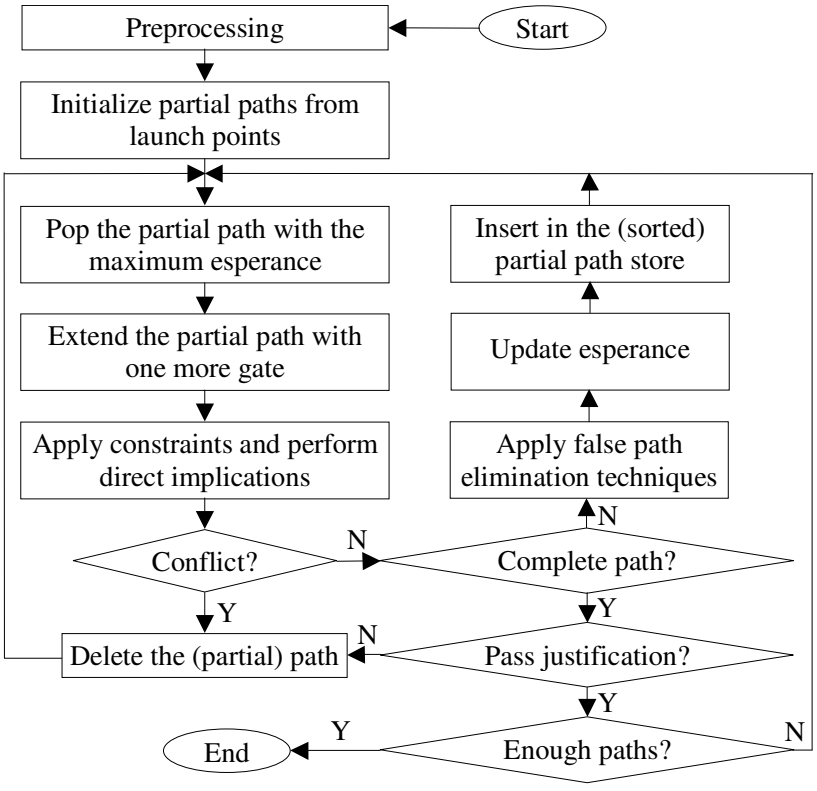


Figure 7. KLPG path generation algorithm.

1.5 Structure of Dissertation

The remainder of the dissertation is organized as follows. Section 2 contains the details of a dynamic compaction algorithm for the generation of compact delay test sets. To speed up the dynamic compaction procedure, in section 3 an improved dynamic compaction algorithm with recursive learning is presented and applied to benchmark circuits and industrial designs. In section 4, a realistic low cost fault coverage metric targeting both global and local delay fault is developed and implemented in the KLPG test generation. In section 5, an improved KLPG test generation flow is applied to an AMD quad-core microprocessor on silicon. Finally, section 6 concludes the dissertation with future directions.

2. DYNAMIC COMPACTION FOR COMPACT DELAY TEST GENERATION

2.1 Motivation

The path delay fault model [17] is used to detect distributed and small delay defects in integrated circuits. The challenge of the path delay fault model is that the number of paths is exponential in the circuit size. One strategy is to target a subset of paths which contains at least one of the longest testable paths passing through each line or gate [14][28][29][33][34][35][36]. More recently, small delay defect tools have been built on top of the transition fault framework, to providing timing information to guide the test generation towards selecting longer paths. But commercial SDD tools produce very high pattern counts.

In [28], an efficient automatic test pattern generation (ATPG) algorithm was developed to test the K Longest Paths Per Gate (KLPG) in a combinational circuit and extended to sequential circuits in [29]. A fault coverage metric was developed to show the theoretical high quality of KLPG [37] and the benefits were demonstrated on silicon [38]. The primary barrier to the use of KLPG patterns has been the high pattern count. The existing *CodGen* ATPG tool [28][29] uses greedy, forward-order static compaction. In order to reduce the pattern count and test cost, this section proposes a new dynamic compaction algorithm for generating compacted test sets for *K* longest paths per gate (KLPG) in combinational circuits or scan-based sequential circuits**.**

2.2 Previous Work

For scan-based very large scale integrated (VLSI) circuits, test cost is determined by test application time of a set of test patterns. Test application time is proportional to the length of the scan chains and the size of the test set [39]. In addition, a test set that exceeds the tester memory size requires reloading patterns to achieve the desired coverage, which is very expensive.

A test vector or test pattern is composed of a set of values on all primary inputs (for a combinational circuit), and all scan flip-flop cells (for a sequential circuit). Automatic test pattern generation (ATPG) tool will assign binary values (0 or 1) on a subset of primary inputs (PI's) and scan flip-flop cells, in order to detect targeted faults. The remaining values are don't care (X). Test compaction includes techniques to reduce test size by merging test patterns with non-conflicting values [40]. In general, test compaction techniques can be classified as *static* compaction or *dynamic* compaction. Static compaction techniques are performed after test sets have already been generated, while dynamic compaction techniques are integrated into the test generation process. Many compaction algorithms have been proposed in the literature for test compaction in combinational and fully-scanned sequential circuits. The following two sections address static and dynamic compaction processes, respectively.

2.2.1 Static Compaction

Static compaction [41] is also called post-generation compaction, which is independent of the test generation process. It can be applied to any set of test vectors to

reduce the test size. Even if dynamic compaction is performed during test generation, static compaction can be used to further reduce the test size.

For static compaction, two test vectors are compactable if every bit is compatible. If the same bit in both vectors is assigned to the same logic value ("0" or "1"), or it is a don't care value ("X") in at least one of them, these two vectors are compatible and can be merged. The concept of static compaction is illustrated using a simple example in Figure 8. As shown in Figure 8, every bit in 8-bit vectors *V1* and *V2* is either assigned to the same logic value or there is "X" in at least one of them. Thus *V1* and *V2* can be compacted together to form a new vector *V3*.

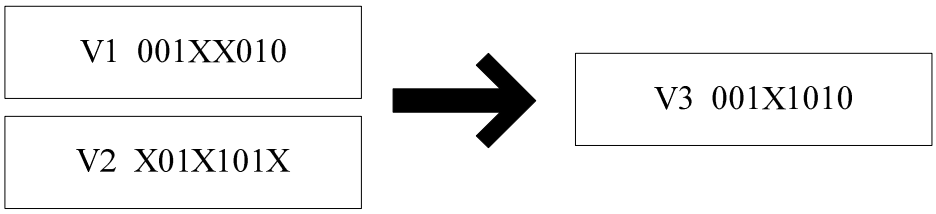


Figure 8. Static compaction of two 8-bit vectors.

Many static compaction techniques [42][43][44][45][46] have been proposed to reduce pattern count without reducing fault coverage. For KLPG test, a forward order greedy static compaction scheme is used to reduce the test size [28]. As shown in Figure 9 [47], every new pattern is compared against patterns in the compacted pattern list in order and is always merged with the first compatible one. The test size produced by the

forward greedy algorithm is only slightly larger than the near-optimal results produced by a simulated annealing algorithm [48].

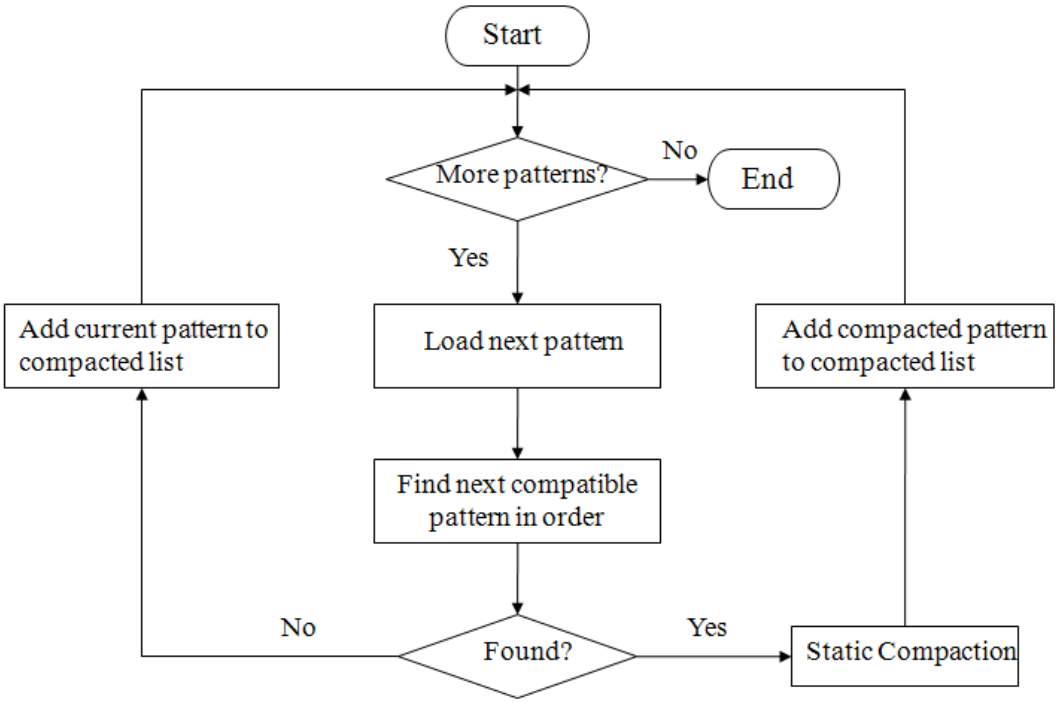


Figure 9. Greedy static compaction flow.

2.2.2 Dynamic Compaction

Dynamic compaction [42] is performed during test generation, and can achieve greater reduction in pattern count than static compaction. Many dynamic compaction methods [49][50][51][52] aim at maximizing the number of stuck-at faults detected by a test pattern. The classic approach is to generate a pattern for one fault, and then use heuristics to modify the unspecified bits, and drop other detected faults in the fault list

via fault simulation. This approach does not work well for path delay test due to the low fortuitous detection rate. Several compaction techniques targeting path delay faults have been proposed [53][54][55]. In [53], maximal compatible path delay fault sets are first derived based on a six-valued algebra and then a test is generated for faults in each of these sets. The memory requirements depend on the number of paths and the number of lines in a path. The approaches in [54][55] try to simultaneously test paths with crossing points so as to fortuitously detect many faults which may not be included in the target fault list. This is similar to fortuitously increasing K in KLPG, but does not explicitly compact compatible paths and does not guarantee the robustness or non-robustness of the fortuitously detected paths. In addition, most of these techniques require a target path fault list and path structure information provided in advance, have high memory consumption for large circuits, have high CPU time complexity, or are difficult to incorporate into the KLPG ATPG algorithm.

Care bit density is the number of (*determined bits* / *total bits*) in test patterns. Normally the average care bit density of the transition fault test is significantly higher than path delay test, particularly for the initial test patterns. Experiments on an industrial design [38] showed the average care bit density of the transition fault test without random fill was 4.59%, while it was 2.23% for the path delay test after static compaction. This low care bit density provides room for dynamic compaction to improve over static compaction.

2.3 Proposed Dynamic Compaction Algorithm

For a given fault, *necessary assignments* (NA) are all the values on circuit lines necessary for the detection of the fault. Necessary assignments include values to activate the fault and propagate its effect to a primary output or a capture scan cell. In this work, we have developed a dynamic compaction approach that compacts paths together based on their necessary assignments, without fault simulation. Rather than working on one pattern at a time, the algorithm considers a pool of paths that are currently being compacted into a set of patterns. Each new path generated is compared against this path pool. This algorithm was incorporated into the KLPG algorithm and significantly reduces pattern count (up to 4x compared to static compaction) without coverage loss.

When a path is generated and passes final justification, a set of necessary assignments are identified that are necessary to sensitize and propagate the fault along the path. Assignments generated during final justification are not saved, since they may not be necessary.

Consider the following example: we have a complete path *Path1* with falling transition through line *A* with necessary assignments (circles) as shown in Figure 10. Vector pairs (X101XX, X1X0X1) or (1X01XX, X100X1) can test the path. Suppose we have another complete path *Path2* with rising transition through line *B*, with a set of necessary assignments (Xs) as shown in Figure 11, and only vector (X0X1X0, X10XX1) can test it. The necessary assignments for *Path1* and *Path2* are compatible (no conflict in value assignments). In our prior work [28][29] we used a PODEM-like final justification procedure to find a vector pair for each path separately, followed by static compaction.

Due to the intrinsic property of backtracing in the PODEM [32], which backtraces based on the line controllability, we get vector pair (X101XX, X1X0X1) for *Path1* and (X0X1X0, X10XX1) for *Path2*. The vector pairs cannot be compacted together. In fact, *Path1* and *Path2* can be compacted together and tested via vector pair (1001X0, X100X1). If we keep only necessary assignments rather than a vector pair for each complete path, we can combine two sets of necessary assignments together and then apply final justification, which will generate one vector pair (1001X0, X100X1) for both paths, as shown in Figure 12.

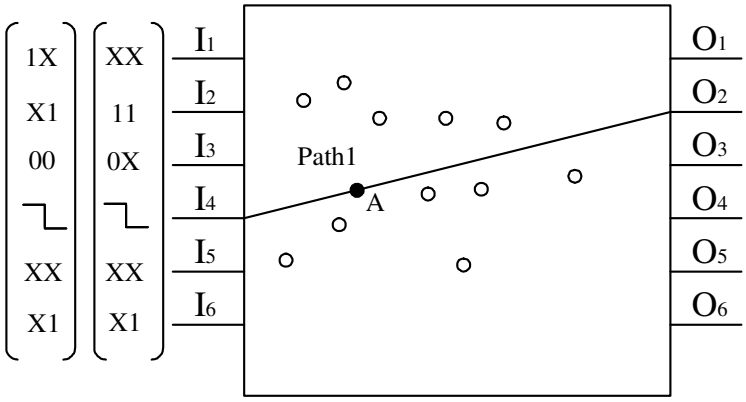


Figure 10. Vector pair and necessary assignments (circles) for Path1.

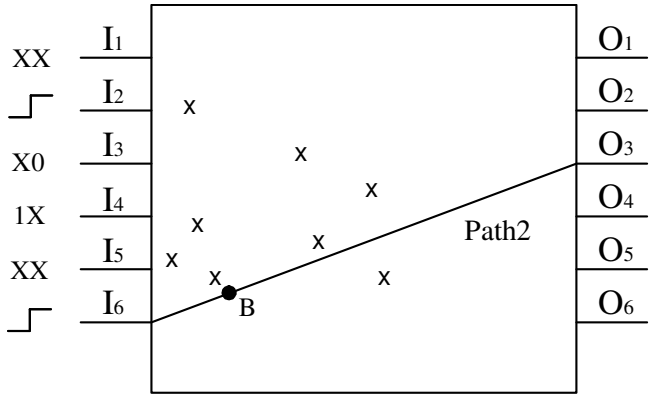


Figure 11. Vector pair and necessary assignments (Xs) for Path2.

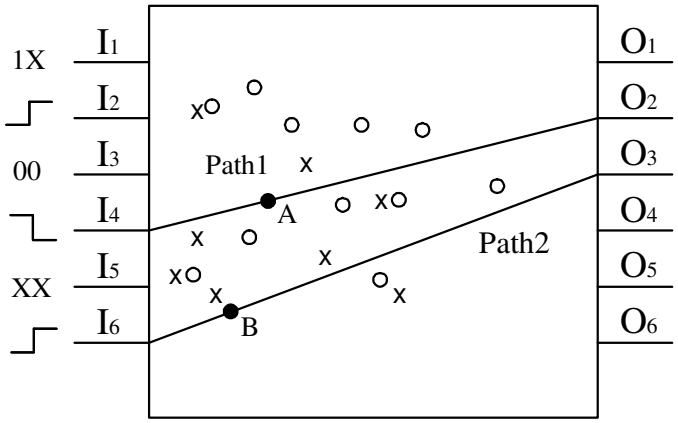


Figure 12. Vector pair and necessary assignments for Path1 and Path2.

Based on the previous example, our dynamic compaction algorithm checks the compatibility between necessary assignments, greatly expanding the compaction space without loss of fault coverage. The approach is to generate the *K* longest rising and falling paths for a line and their necessary assignments, and then compare the necessary assignments for each such path against a set of previously generated paths. The generation of final test vectors is postponed until test generation and dynamic compaction is finished, in order to provide maximum flexibility for compaction.

Procedure *dyn_compact*(*F*, *POOL*), given in Figure 13, describes the details of the dynamic compaction algorithm. It uses a greedy approach, in which each new pattern *F* is compacted with the first compatible pattern in *POOL*. *POOL* is a data structure created to save patterns. Due to memory limitations, we cannot save all patterns in *POOL*. So the size of *POOL* is set to *N*, which means at most *N* patterns can be saved in *POOL*. Patterns in *POOL* are sorted in non-increasing order of the number of necessary assignments in order to compact as many paths as possible into a pattern before it is written out. Our experiments show that non-decreasing order will even out the care bit density and result in a more compact test set. When *POOL* is full and a new pattern is generated, the pattern with the largest number of necessary assignments (the first pattern in *POOL*) is popped. Final justification is then performed on this pattern and its corresponding vector pair is written out. We will show the influence of *POOL* size on compaction results in the experimental results. In the current implementation a greedy forward-order approach is used for locating compatible paths. Based on static compaction results [29], we expect that only a small fraction of patterns must be held in memory in order to achieve good results for dynamic compaction.

Procedure *dyn_compact*(*F*, *POOL*)

1. If *POOL* is empty, insert pattern *F* into *POOL* and return. Otherwise set pointer *P* to the first pattern in *POOL* and go to step 3.
2. Set pointer *P* to the next pattern in *POOL*. If *P* is pointing to empty (the end of *POOL*), go to step 6. Otherwise go to step 3.
3. Do conflict check between *F* and *P*. If there is a conflict, go to step 2. Otherwise go to step 4.
4. Combine two sets of necessary assignments *F* and *P*, and save them as *K*. Check for direct implication conflicts in *K*. If no conflicts, go to step 5. Otherwise delete *K* and go to step 2.
5. Do final justification for *K*. If *K* passes final justification, update *P* by combining necessary assignments of *F* into it and return. Otherwise delete *K* and go to step 2.
6. Insert *F* into *POOL* as a pattern. Return.

Figure 13. Pseudo code of dynamic compaction algorithm.

To help better understand the details, Figure 14 gives the dynamic compaction flowchart.

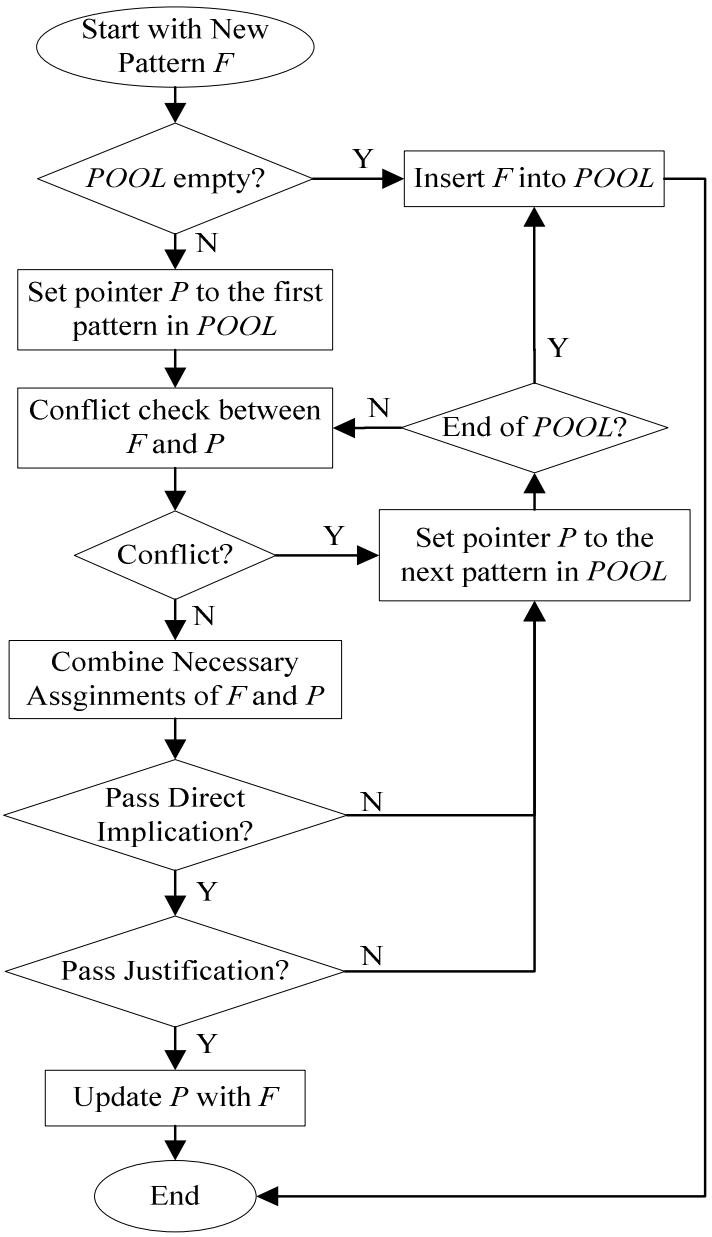


Figure 14. Flowchart of dynamic compaction algorithm.

2.4 Dynamic Compaction for KLPG Test

The proposed dynamic compaction algorithm was integrated into the *CodGen* KLPG ATPG. Procedure *klpg_dc*( ) in Figure 15 describes the test generation flow with dynamic compaction. Here a *pattern* is a set of necessary assignments that can successfully yield a test vector pair. A *vector* contains specific values at launch points (e.g. scan cell outputs).

Procedure *klpg_dc*( )

1. Initialize pattern pool *POOL* as empty.
2. Use KLPG to generate a successful longest path *I* through a line, resulting in pattern *F*. *F* contains all necessary assignment information before justification. Justification of *F* is performed to check that the path is sensitizable, but the resulting primary input values are not stored. If no more paths can be generated or we have enough paths, go to step 4. Otherwise go to step 3.
3. Call procedure *dyn_compact*(*F*,*POOL*). Go to step 2.
4. Do final justification for all patterns in *POOL* one by one to generate the final vectors. Procedure is finished.

Figure 15. Test generation flow with dynamic compaction.

2.5 Dynamic Compaction Experimental Results for KLPG Test

Experiments were conducted to show the advantages of our proposed dynamic compaction algorithm over the static compaction method for KLPG tests. All programs were implemented in C++ and run on a Windows XP PC with Intel Core 2 Duo 6300 (1.86 GHz) processor and 2 GB memory.

We performed experiments on full scan versions of ISCAS89 benchmarks and three industry designs, generating KLPG tests with K=1. Table 1 shows the information of all circuits used in the experiments. The targeted fault space is identical to the transition fault test, which includes every line in the circuit. The number of faults is twice the number of lines in the circuit, since it is assumed that there are both slow-to-rise and slow-to-fall delay faults at each fault site. But the actual number of detectable faults is less than the total number of faults due to the internal and external constraints. Internal constraints include the lines with preset values, such as lines tied to ground/supply voltage. External constraints include the constraints from low-cost ATE. For example, low-cost ATE does not allow observation of the primary outputs and it requires primary inputs to remain unchanged during test vector application. Column 4 lists the number of scan cells for each circuit. There is only one scan chain for all ISCAS89 circuits. The industrial design chip1 contains 4 scan chains and chip2 contains 16 scan chains. Both are partial scan design with embedded memories. Chip2a is a controller module in chip2. The industrial design chip3 is a full scan design with 6 scan chains.

Table 1. Circuits used in experiments.

Circuit	# Lines	# Scan Cells	# Scan Chains	Scan Type
s1423	1 423	74	1	Full
s1488	1 488	6	1	Full
s1494	1 494	6	1	Full
s5378	5 378	179	1	Full
s9234	9 234	211	1	Full
s13207	13 207	638	1	Full
s15850	15 850	534	1	Full
s35932	35 932	1 728	1	Full
s38417	38 417	1 636	1	Full
s38584	38 584	1 426	1	Full
chip1	86 612	3 503	4	Partial
chip2	1 956 942	57 352	16	Partial
chip2a	40 590	14 963	8	Partial
chip3	1 085 052	9 372	16	Full

2.5.1 KLPG Robust Test with Dynamic Compaction

Two test strategies were used in our experiments: launch-on-capture (LOC) and launch-on-shift (LOS). Table 2 shows the results for generating the longest robustly-testable rising and falling paths through each line (K=1), with static and dynamic compaction in LOC mode. The *POOL* size N was set to 1000 for all circuits. Column 1 give the circuit name. Since we generate both longest slow-to-rise and slow-to-fall paths through each line, the number of faults is roughly twice the number of lines. The total number of testable faults is less than the total number of faults because of the LOC and LOS constraints, and the constraints of low-cost testers (fixed inputs and masked outputs). Columns 2-5 give the results under LOC. Column 2 shows the total number of paths generated by KLPG, which is equal to the pattern count without any compaction. Column 3 shows the number of test patterns with static compaction (SC) and dynamic compaction (DC). For example, for s38417, 946 patterns are generated with static compaction, which is reduced to 422 with dynamic compaction. Column 4 shows the percentage pattern reduction. The reduction is small for small circuits, and higher in larger circuits, such as chip1, chip2a and chip3. This is consistent with the lower care bit density of uncompacted patterns in these designs. The last column gives the CPU time. The numbers are times for SC and DC. In most cases DC takes about twice the CPU time of SC.

Table 2. Comparison of KLPG (K=1) test size with static and dynamic compaction (robust test LOC).

Circuit	Launch-on-Capture			
	# Paths	# Patterns (SC/DC)	% Reduction	Time (m:s) (SC/DC)
s1423	397	222/138	38	00:09/00:10
s1488	192	87/67	23	00:01/00:01
s1494	193	86/63	27	00:01/00:01
s5378	1799	407/236	42	00:07/00:16
s9234	2376	790/405	49	1:35/2:02
s13207	3246	900/718	20	00:59/1:35
s15850	2645	471/278	41	1:19/1:42
s35932	9762	36/26	28	5:03/06:54
s38417	14917	946/422	55	07:16/28:00
s38584	9725	519/253	51	4:23/10:32
chip1	14807	2477/1024	59	32:21/72:38
chip2a	7019	1877/631	66	199:54/228:00
chip3	47822	18203/5962	67	16.7hrs/33.4hrs

Table 3 shows the results for generating the longest robustly-testable rising and falling paths through each line, with static and dynamic compaction in LOS mode. Similar pattern reduction rate is achieved for LOS mode.

Table 3. Comparison of KLPG (K=1) test size with static and dynamic compaction (robust test LOS).

Circuit	Launch-on-Shift			
	# Paths	# Patterns (SC/DC)	% Reduction	Time (m:s) (SC/DC)
s1423	701	197/123	39	00:04/00:06
s1488	206	78/64	18	00:01/00:01
s1494	204	79/64	19	00:01/00:01
s5378	1112	84/49	42	00.03/00:06
s9234	3649	710/463	35	01:03/01:46
s13207	6843	1624/1421	12	00:34/01:56
s15850	5833	645/307	52	00:50/02:24
s35932	12194	44/35	20	03:09/07:43
s38417	17665	656/357	46	02:03/16:07
s38584	21135	683/592	13	02:55/20:59
chip1	20139	858/467	46	44:40/86:36
chip2a	10512	2829/1140	60	38:48/57:26
chip3	58154	9760/3848	61	3.9hrs/6.8hrs

2.5.2 KLPG Non-Robust Test with Dynamic Compaction

Tables 4 and 5 show the compaction results for the longest rising and falling non-robustly-testable paths through each line, with SC and DC under LOC and LOS respectively. The pattern count reduction and CPU time increase are similar to the results for robust paths.

Table 4. Comparison of KLPG (K=1) test size with static and dynamic compaction (non-robust test LOC).

Circuit	Launch-on-Capture			
	# Paths	# Patterns (SC/DC)	% Reduction	Time (m:s) (SC/DC)
s1423	755	333/147	56	00:21/00:22
s1488	531	144/102	29	00:02/00:02
s1494	537	143/102	29	00:02/00:02
s5378	2428	362/213	41	00:17/00:28
s9234	3806	949/422	56	05:40/06:20
s13207	5674	629/416	34	05:06/05:54
s15850	4931	480/223	54	05:21/06:13
s35932	14569	36/25	31	14:03/17:15
s38417	26390	1109/414	63	16:32/50:20
s38584	17815	1177/391	67	12:40/27:22
chip1	27371	2866/1197	58	100:47/188:01
chip2a	15038	2752/1116	59	18.7hrs/20.2hrs
chip3	78432	23096/7540	67	37.7/64.3hrs

Table 5. Comparison of KLPG (K=1) test size with static and dynamic compaction (non-robust test LOS).

Circuit	Launch-on-Shift			
	# Paths	# Patterns (SC/DC)	% Reduction	Time (m:s) (SC/DC)
s1423	1123	267/162	39	00:11/00:13
s1488	425	97/91	6	00.01/00:01
s1494	431	98/91	7	00.01/00:01
s5378	2028	226/151	33	00:09/00:11
s9234	5257	578/389	33	03:03/03:45
s13207	8981	786/616	22	01:13/02:28
s15850	8460	445/235	47	01:17/02:37
s35932	18489	36/29	19	07:40/13:08
s38417	26832	718/288	60	06:29/24:25
s38584	28059	450/337	25	06:59/27:11
chip1	38766	1389/739	47	91:35/188:30
chip2a	25093	4939/3032	39	140:07/206:16
chip3	101377	14574/6288	57	6.7hrs/12.4hrs

2.5.3 Pool Size Influence on Dynamic Compaction

In order to analyze the influence of the *POOL* size *N* on our dynamic compaction algorithm, we vary *N* while generating robust paths under LOC, as shown in Table 6. Columns 2-3, 4-5, 6-7 and 8-9 give the pattern count and CPU time for *N* with 100, 200, 500 and 1000 respectively. Increasing *N* reduces pattern count at the expense of more CPU time. The ISCAS89 circuits are saturated at *N*=500, but the three industry circuits

see a significant benefit for *N*=1000. But for all circuits there is a diminishing return. For example, for chip1, 1264 patterns are generated under *N*=500 and 1024 patterns are generated under *N*=1000. Diminishing returns can be explained by the phenomenon that one pattern can be compacted to many other patterns. Not compacting a pattern into a previously written-out pattern will not influence the chance of compacting it into another pattern in the *POOL*. For chip3, pattern count is big under *N*=100 and *N*=200, while 6546 patterns are generated under *N*=500 and 5962 patterns are generated under *N*=1000. When further increasing *N* to 2000, 4412 patterns are generated while CPU time increases to 36 hrs. Ideally infinite pool size *N* will achieve the best results. But some tradeoff must be made in practice due to the limitation of computer memory size and running time. A moderate *N* has been enough to achieve good results from experiments.

The efficiency of the greedy approach was also checked by changing the pattern order in *POOL* for a specific *N*. Both forward order and backward order greedy dynamic compaction algorithms are implemented. Results show that the pattern count remains similar for different pattern orders. This is consistent with our theory that our greedy dynamic compaction approach can achieve good results while holding a small fraction of paths in memory.

Table 6. POOL size influence on compaction (K=1 robust test).

Circuit	N=100		N=200		N=500		N=1000	
	# Patterns	CPU Time (m:s)	# Patterns	CPU Time (m:s)	# Patterns	CPU Time (m:s)	# Patterns	CPU Time (m:s)
s1423	141	00:09	140	00:09	140	00:09	140	00:09
s1488	67	00:01	67	00:01	67	00:01	67	00:01
s1494	63	00:01	63	00:01	63	00:01	63	00:01
s5378	241	00:11	231	00:14	231	00:14	231	00:14
s9234	533	01:32	423	01:39	408	01:51	408	01:51
s13207	752	01:04	731	01:15	719	01:53	719	01:28
s15850	303	01:33	290	01:41	289	01:38	289	01:38
s35932	24	06:36	24	06:36	24	06:36	24	06:36
s38417	476	14:30	442	20:01	425	25:57	425	25:57
s38584	281	07:14	253	09:03	249	09:10	249	09:10
chip1	1703	80:14	1552	84:22	1264	100:37	1201	123:24
chip2a	1102	192:49	988	199:43	716	213:49	669	208:37
chip3	10521	20.5hrs	9161	24.8hrs	6456	26hrs	5431	29.4hrs

2.5.4 Pattern Count Comparison with Commercial Tool

A. KLPG-1 Test

To test path delay faults, two-time-frame vectors are required. Path delay faults are classified into several groups according to the different sensitization criteria [31].

Robust sensitization criterion [56] allows the fault on the target path to be observed independent of the delays on signals outside the target path. In other words, the slow signal is able to propagate through the robust testable path independent of the

delays on the side inputs to the path. Figure 16 illustrates the robust propagation criterion for an OR gate. In this example, *a* is the on-path input and *b* is the off-path input. The two values on inputs denote values under vector pair *(v1, v2)*. When there is a falling transition at input *a*, it requires *b* to be unspecified (X) for vector *v1* and be 0 for vector v2. Thus the fault effect on *a* will be propagated to the output signal of the OR gate regardless of the fault on side input *b*. Similarly, when there is a rising transition on input *a*, it requires *b* to be a stable non-controlling value 0. So the fault on the target path will always be observable at the output.

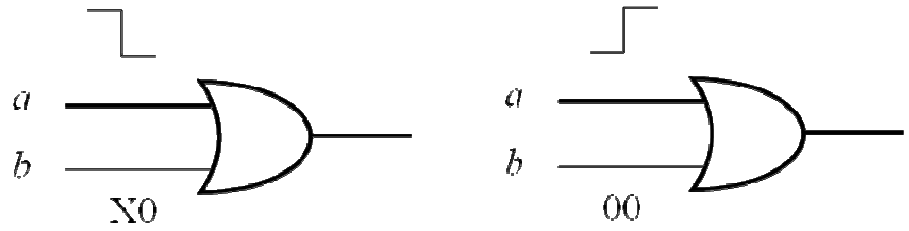


Figure 16. Robust sensitization criterion for OR gate.

Non-robust sensitization criterion [56] is less stringent than the robust sensitization criterion. The fault detection through a non-robust testable path is dependent on the delays outside the target path, such as on the signal arrival times at the side inputs.

A test that guarantees the detection of a path delay fault, when no other delay fault is present, is called a non-robust test for that path. Consider the OR gate in Figure 17. If there is a rising transition on the on-path input *a* and a falling transition on off-path input *b*, the transition on the output of the OR gate depends on the arrival time of the

input transitions. If the falling transition on off-path input *b* occurs later than the rising transition on on-path input *a*, it will mask the fault effect from *a* to the output. If the falling transition on off-path *b* happens earlier than the rising transitions on on-path input *a*, the fault effect on on-path input *a* is still observable at the output.

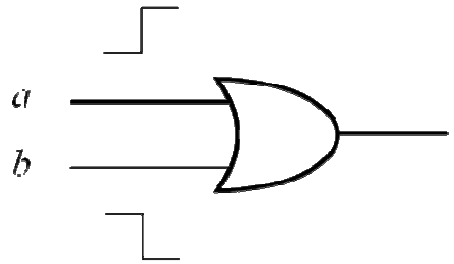


Figure 17. Non-robust sensitization criterion for OR gate.

Functional sensitization criterion [57] further releases the constraints compared to the non-robust sensitization criterion. The detection of faults also depends on the delays outside the target path. Furthermore, in order to detect the target fault, the functional sensitization criterion requires that multiple faulty paths exist in the circuit. Figure 18 illustrates the functional sensitization criterion for an OR gate. When both on-path input *a* and off-path input *b* have rising transitions, it requires both transitions to be late in order to propagate the fault at *a* to the output of the OR gate, since the arrival time of the output signal is determined by the earlier of the two rising transitions at the OR gate.

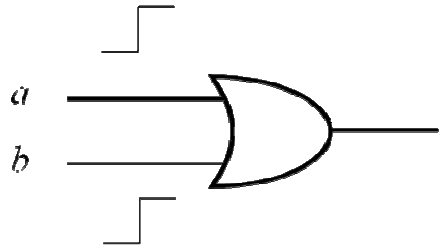


Figure 18. Functional sensitization criterion for OR gate.

Figure 19 shows the test composition of the KLPG-1 test. A KLPG-1 test set, consisting of the robust longest rising and falling path through each line, topped off with non-robust KLPG patterns, topped off with long transition fault patterns, achieves the same transition fault coverage as a transition fault test set, but with higher quality, since it targets the same fault space and smaller delay defects. Long transition fault test has higher quality than the traditional transition fault test because the traditional one assumes large local delay and propagates the fault through any path (usually a short path). In our test generation, this case usually happens when the local delay fault can only be activated or propagated through multiple paths with functional sensitization criterion. Thus the test quality is determined by the length of the shortest path in the activating or propagating path set. The longer the shortest path, the smaller the local delay fault that can be detected. The best transition fault test, in terms of the detected local delay fault size, cannot be guaranteed to be generated by our tool but it should be better than the traditional transition fault test.

The longest non-robust test for a fault may be longer than the longest robust test for the fault. However, in KLPG-1, priority is given to the certainty of robust detection. Detection reliability and testing the longest paths can be achieved by generating non-

robust tests for all faults that already have a robust test, and keeping the non-robust test, in addition to the robust test, if it tests a longer path. Due to the pattern count increase, we did not pursue this approach.

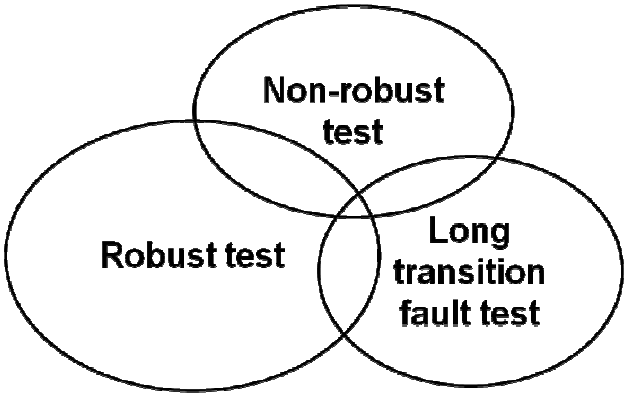


Figure 19. Test composition of KLPG-1.

B. KLPG-1 vs. Commercial Tool

The drawback of a KLPG-1 test set has been increased pattern count. Table 7 shows the test size comparison between dynamically compacted KLPG-1 test sets compared to dynamically compacted transition fault patterns generated by a commercial tool. Both *CodGen* and the commercial tool use launch-on-capture mode. *POOL* size is fixed at 3000 for KLPG-1 test.

As can be seen, our dynamically compacted KLPG-1 test sets are similar to and in several cases (such as for s38584, chip1 and chip2a) smaller than the commercial transition fault test sets. This is quite promising considering the higher quality of the

KLPG test patterns and the maturity of the commercial tool compared to our university tool. However, the KLPG-1 test is still significantly larger than the TF test for chip3.

Table 7. KLPG-1 vs. commercial tool.

Circuit	KLPG Robust	KLPG Non-robust	Long Transition	Robust +Non-robust + Long TF	Commercial Tool
s1423	140	19	30	189	95
s1488	67	45	2	114	102
s1494	63	50	2	115	101
s5378	231	28	0	259	194
s9234	408	41	5	454	465
s13207	719	11	72	802	382
s15850	289	6	7	302	231
s35932	24	4	0	28	68
s38417	425	41	1	467	365
s38584	249	134	70	453	528
chip1	1201	489	157	1853	1900
chip2a	630	438	777	1845	2537
chip3	4077	1422	538	6037	1445

2.6 Experimental Results for Transition Fault Test and Stuck-at Test

The proposed dynamic compaction algorithm is generic in nature so it can be applied to the test generation of any kind test. This algorithm has been extended to deal with transition fault test and stuck-at fault test.

2.6.1 Dynamic Compaction for Transition Fault Test

CodGen has been improved to generate transition fault tests. Figure 20 shows the transition fault test generation flow. It inherits the framework from KLPG test. The only difference is in the partial path extension stage. For transition fault test, in every iteration of path generation, the partial path with the smallest max esperance is popped from the sorted path store and extended by adding one fanout gate with the smallest max esperance. In contrast, for KLPG test, the partial path with the largest max esperance is always popped first and extended to the fanout gate with the largest max experance. Thus the paths generated for transition fault test are normally the shortest ones, which contain fewer necessary assignments and are easier to compact.

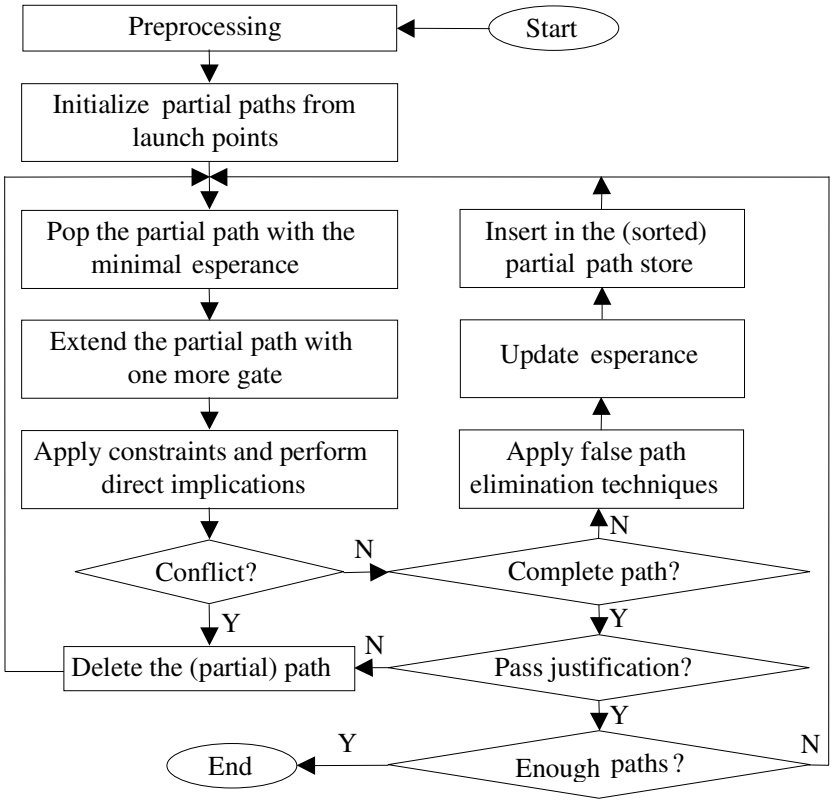


Figure 20. Transition fault test generation algorithm.

Table 8 compares the transition fault test results generated by modified *CodGen* with KLPG-1 test. Both KLPG-1 and transition fault (TF) test are in launch-on-capture mode. Pool size is set as 3000 for both tests. Column 2 shows the number of paths generated under two tests. To achieve the same transition fault coverage as KLPG-1 test, normally more paths are generated for TF test. This is because a longer KLPG-1 path through a given fault site normally covers more transition faults than a short transition fault path. Column 3 lists the pattern count for TF, KLPG-1 and the commercial TF tool. For most cases, the TF test pattern count is much less than KLPG-1, since KLPG-1

patterns have higher care bit density than TF test, which reduces the space for compaction. One special case is s35932, for which the TF test pattern count is higher than KLPG-1 test. This is because s35932 has many near-critical TF test paths with similar care bit density to KLPG-1 paths. Column 4 lists the average path length for the two tests. The data verifies our assumption that most transition fault paths are shorter than KLPG-1 paths. For example, the average path length of TF test is 8.77 compared to 13.49 (54% higher) for KLPG-1 paths. For special case s35932, the average path length of TF test is 12.63 compared to 12.79 for KLPG-1 test. This indicates that most TF test paths have similar length to KLPG-1 paths and explains why the TF test pattern count is higher than KLPG-1 test for s35932. CPU time is listed in the last column. TF test generation takes much less time than KLPG-1 test. This is because most TF test paths are shorter and so easier to generate compared to KLPG-1 paths. In all circuits except s1488 and s1494, our TF test sets with dynamic compaction are significantly smaller than produced by the commercial TF tool, albeit using significantly more CPU time.

Table 8. TF vs KLPG-1.

.Circuit	Launch-on-Capture			
	# Paths (TF/KLPG-1)	# Patterns (TF/KLPG-1 /Comm)	Avg Path Length (TF/KLPG-1)	Time (m:s) (TF/KLPG-1)
s1423	791/744	68/189/95	7.11/17.36	00:09/00:32
s1488	543/535	103/114/102	10.67/10.98	00:07/00:07
s1494	549/537	103/115/101	10.70/10.98	00:07/00:07
s5378	2435/1964	184/259/194	9.88/11.78	00:28/00:35
s9234	4086/3399	230/454/465	17.01/19.73	03:08/06:29
s13207	6089/5450	356/802/382	16.04/19.6	03:27/03:58
s15850	4940/4546	134/302/231	13.88/18.83	03:48/07:17
s35932	14537/11250	35/28/68	12.63/12.79	14:34/11:04
s38417	25327/21855	254/467/365	14.10/18.14	38:32/43:51
s38584	18923/17889	320/453/528	8.39/10.27	23:55/27:36
chip1	31896/28229	1600/1853/1900	8.77/13.49	200:26/453:58
chip2a	14070/16128	1207/1845/2375	12.26/16.96	14.3hrs/25.5hrs
chip3	81334/69740	1067/6037/1445	10.53/23.55	23.6hrs/80.7hrs

2.6.2 Dynamic Compaction for Stuck-at Fault Test

In [58], a scalable dynamic compaction technique was recently integrated into a D-algorithm [59][60][61] based ATPG engine to generate compact test pattern sets for stuck-at faults. This technique inherits the idea of our dynamic compaction algorithm and uses necessary assignments as guidance to accommodate detections of more faults by the same test vector. The guidance is based on a preprocessing step that computes sets of compatible faults with their necessary assignments. This approach generates

minimal or close to minimal test sets [51] for ISCAS85 circuits. For industrial circuits, it achieves smaller test sets than other available methods, at reasonable CPU cost.

2.7 Conclusions

We have proposed a new dynamic compaction algorithm [62] for generating compacted test sets for *K* longest paths per gate (KLPG) in combinational circuits or scan-based sequential circuits. This algorithm uses a greedy approach to compact paths with non-conflicting assignments together during test generation. Experimental results for ISCAS89 benchmark circuits and three industry circuits show that the pattern count of KLPG can be significantly reduced (up to 4x compared to static compaction) using the proposed method. The pattern count after dynamic compaction is comparable to the number of transition fault tests, while achieving higher test quality. This algorithm is also a generic algorithm and also achieves significant test size reduction for transition fault test and stuck-at fault test.

3. IMPROVED DYNAMIC COMPACTION WITH RECURSIVE LEARNING

3.1 Motivation

A dynamic compaction approach has been proposed in section 2 to compact paths together based on their necessary assignments [62]. As each path is generated, its necessary assignments are checked against a pool of test patterns. If the necessary assignments of the path are compatible with those in a pattern, an attempt is made to justify a vector pair to apply the test, using a PODEM-like algorithm. This compaction algorithm significantly reduces KLPG pattern count (up to 4x compared to static compaction) without coverage loss. But an analysis of the dynamic compaction algorithm shows that the failure rate of final justification ranges from 0.5% (s35932) to 98% (chip3), relatively independent of the pool size. In other words, necessary assignment compatibility is not sufficient to screen for conflicting paths. Table 9 presents the final justification failure rate with dynamic compaction in launch-on capture mode with Pool size 1000. For the three industrial circuits, the final justification failure rate is high (up to 98% for chip3). The final justification failures are due to either no solution or algorithm abort. If there is any value conflict in the circuit, final justification has no solution. If final justification algorithm hits the preset algorithm backtrace limit, it gives up. An analysis shows that most final justification failures in dynamic compaction are due to value conflicts. This indicates that necessary assignments are not sufficient to filter out incompatible paths in several designs before entering final justification

procedure. Since each execution of final justification is expensive, this high failure rate leads to a significant CPU time increase over static compaction. In order to reduce the CPU time and accelerate dynamic compaction, we must find more necessary assignments to filter incompatible cases prior to final justification.

Table 9. Final justification failure rate.

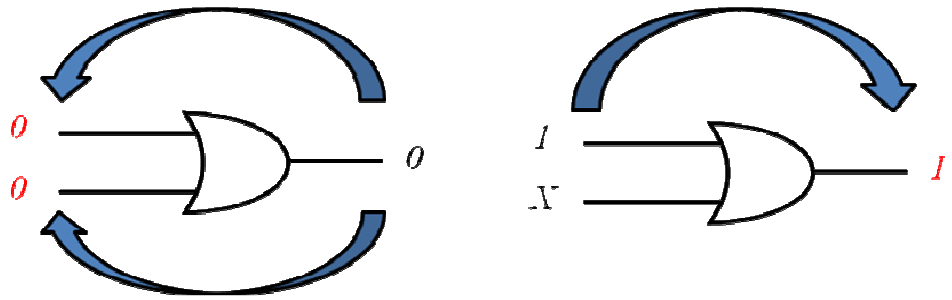
Circuit	Final Justification Failure Rate (%)
s1423	13.10%
s1488	12%
s5378	56.60%
s9234	20.50%
s13207	6.50%
s15850	5.40%
s35932	0.50%
s38417	44.90%
s38584	20.70%
chip1	82.10%
chip2a	82.70%
chip3	98%

This section proposes an improved dynamic compaction algorithm with recursive learning, which tries to explore more indirect necessary assignments in the circuit given a set of line values, in order to trim the search space prior to final justification and speed up the dynamic compaction procedure. Results show that the improved compaction algorithm is effective in ISCAS89 and three industry circuits. The failure rate of final

justification during dynamic compaction can be significantly reduced using the proposed method.

3.2 Previous Work

In standard ATPG, *direct implication* [28] is used to discover *necessary assignments* (values assigned to lines). A direct implication on a gate is one where the input or output of that gate can be uniquely determined from other values assigned to that gate. Direct implication can be classified as forward implication or backward implication. Figure 21(a) is an example of backward implication. Given a 0 on the output of an OR gate, it is only true when both inputs are assigned the value 0. Figure 21(b) shows an example of forward implication. Direct implication can be seen as the logical consequence of the truth table for a logic function. An *indirect implication* can not be directly derived from the logic function truth table but is caused by the circuit structure, which is explored through learning.



(a) (b)

Figure 21. Direct implication examples.

Learning was first introduced in [43][63]. In [43], a static learning procedure is performed during the preprocessing phase to explore global implications, i.e., once for the whole circuit. Then an improved implication procedure to apply the learning dynamically was proposed in [63], which is performed for each branching step. Oriented dynamic learning was introduced in [64], which performs learning only at a small subset of the signals used by [43][63]. But the approaches in [43][63][64] do not guarantee that all necessary assignments can be identified. Later recursive learning [65] was proposed for test generation, design verification, and redundancy identification. It is a general method in the sense that it is not restricted to any logic alphabet and can be called recursively to find all necessary assignments.

Temporary necessary assignments are injected at arbitrary signals in the circuit during learning to explore the common logic consequences. Figure 22 shows an example of a level 1 recursive learning procedure. Assume F=0 is the only assignment available. Direct implications cannot be derived from F=0. However, there exists the indirect implication that F=0 => C=0. Beginning at the point when direct implications cannot be made (F=0), temporary assignments are injected into the circuit for each choice of assignments that would justify the current node value. The value F=0 makes the gate *G1* unjustified. We enter level 1 recursive learning with two possible justifications for *G1*: J_1 = {D=0, E=x} and J_2 = {D=x, E=0}. For {D=0, E=x}, A=C=0 is derived through direct implication at *G2*. For {D=x, E=0}, B=C=0 is derived through direct implication at *G3*. In Figure 22, two possible justifications and their necessary assignments are marked with quote and double quotes. Since C=0 is the intersection of the necessary

assignments for both justifications, J_1 and J_2, it becomes a common necessary assignment for *F*=0. If *C* is further driven by an AND gate, then the learning procedure can be recursively called to explore further implications.

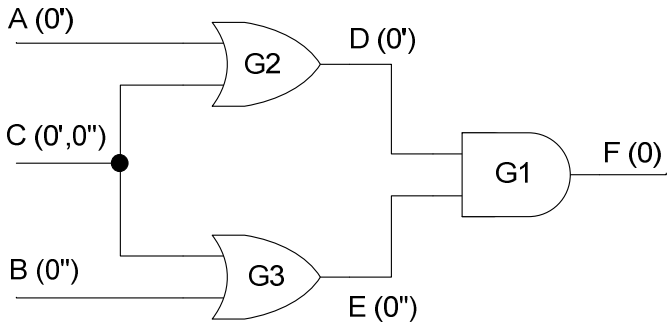


Figure 22. Example of recursive learning.

Figure 23 gives the details of the recursive learning algorithm [65]. The time complexity of recursive learning is exponential in *rmax*, the maximum depth of recursion, but memory grows linearly with *rmax*.

Initialize r=0

Function *make_all_implications*(r,rmax)

make all direct implications and set up a list of U_r of all unjustified gates.

if (r < **rmax)**

for each gate G_i in U_r

set up a list of justifications $C_r^{G_i}$

for each justification J_i in $C_r^{G_i}$

make the assignments in J_i

***make_all_implications*(r+1,rmax)**

if there is one or several signals f in the circuit, which assume the same logic value V for all consistent justifications J_i in $C_r^{G_i}$ then learn:

f=V is uniquely determined in level r

make direct implications for all f in level r

if all justifications are inconsistent, then learn:

given value assignments in level r is inconsistent

Figure 23. Recursive learning algorithm.

3.3 Improved Dynamic Compaction with Recursive Learning

We found that in some circuits, the final justification step during dynamic compaction fails at a high rate, leading to high CPU time. To make our dynamic compaction approach practical for industrial use, we must drastically reduce CPU time. Recursive learning is called whenever a new path is generated or a pattern in POOL is updated, in order to identify more necessary assignments for each path, so that path conflicts can be more accurately identified with necessary assignments. These necessary assignments also reduce the search effort required in final justification. Figure 24 shows the details of the improved dynamic compaction with recursive learning. Large recursion depth is in not practical because CPU time is exponential in depth. However, since it is most likely that unknown necessary assignments must lie in the "logic neighborhood" of the known necessary assignments, it can be expected that the maximum recursion depth to determine all necessary assignments is relatively low [65]. We use a maximum depth of 3. In order to further limit CPU time, recursive learning is only done on patterns with fewer than 25 compacted paths. In addition, patterns that fail compaction more than a certain number of times in a row, such as 500, are written out from *POOL* to speed up the test generation, since those patterns are essentially at their compaction limit.

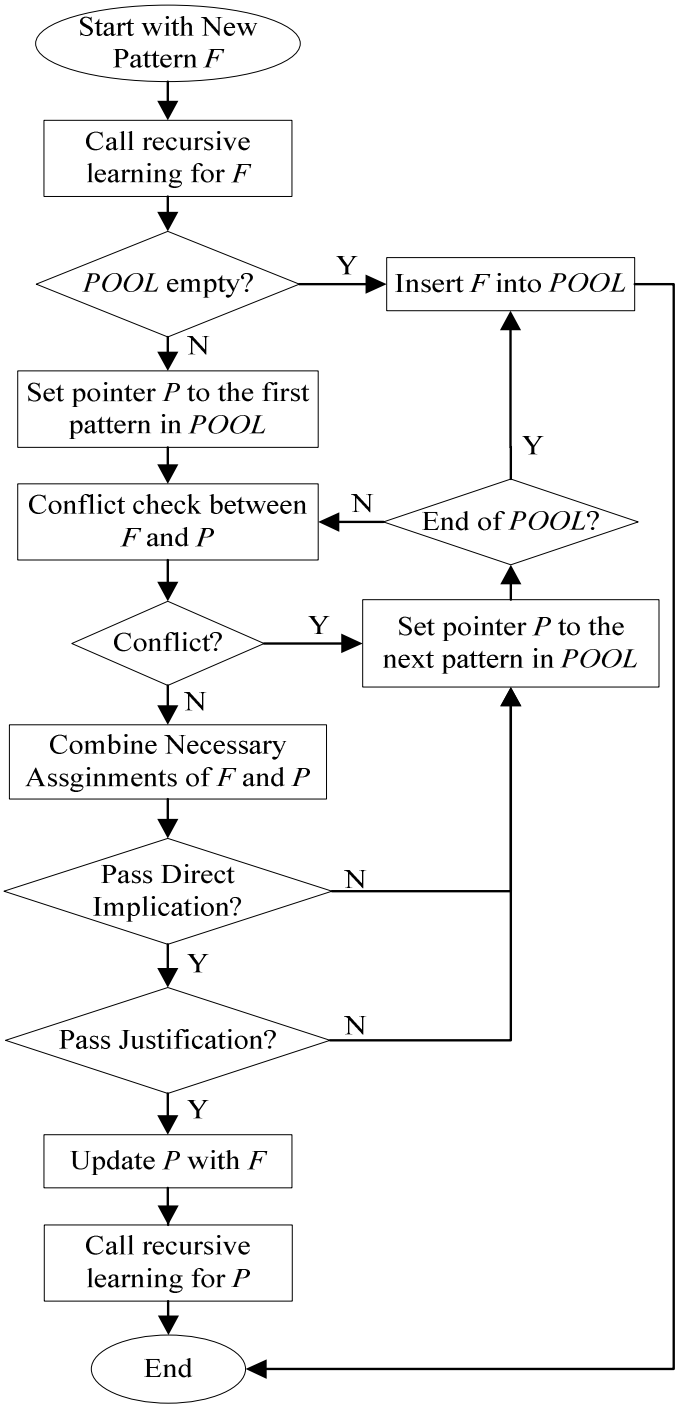


Figure 24. Improved dynamic compaction algorithm.

3.4 Experimental Results

Table 10 gives the results of improved dynamic compaction algorithm with recursive learning for launch-on-capture robust test. Columns 1 give the circuit name and the number of lines. Column 2 gives the total number of test patterns after dynamic compaction without recursive learning (learning depth=0), with recursive learning depth 2 and with learning depth 3 respectively. Column 3 shows the number of total successful final justifications/failed final justifications without recursive learning, with learning depth 2 and with learning depth 3. Column 4 presents the final justification failure rate without recursive learning, with learning depth 2 and with learning depth 3. The CPU time under the three different maximum recursion depths is given in column 5. Normally the failure rate of final justification is greatly reduced with an increase of learning depth, with similar pattern count. For example, for s38584, the failure rate without recursive learning is 20.7% with 426 total patterns. This decreases to 2.1% and 426 patterns with learning depth 2, and further to 1.5% and 422 patterns with learning depth 3. There is not much CPU time decrease for the small circuits in Table 1. But for chip3, a design with ~600k gates, the reduction of final justification failure rate reduces CPU time from 79 hrs (without recursive learning) to 29 hrs (learning depth 2), and to 50 hrs (learning depth 3). Since the CPU time for recursive learning is exponential in the maximum recursion depth, 22 more hours are spent on recursive learning with learning depth 3 than with learning depth 2 for chip3. A tradeoff must be made between CPU time and learning depth in practical applications. In some cases, such as s15850, the final justification failure rate increases with higher learning depth. This is because the

PODEM-like final justification algorithm struggles with the large number of necessary assignments in highly-compacted test patterns, which causes final justification hit the backtrack limit and abort.

Table 10. Improved dynamic compaction with recursive learning (LOC robust test, POOL = 1000).

Circuit	# Patterns (0/2/3)	[#Success/#Fail] (0/2/3)	% failure rate (0/2/3)	Time (m:s) (0/2/3)
s1423	139/138/138	[258/39]/ [259/12] /[259/7]	13.1/4.4/2.6	00:09/00:08/00:10
s1488	67/67/67	[125/17]/[125/6] /[125/3]	12.0/4.6/2.3	00:02/00:02/00:04
s5378	236/230/230	[1563/2037]/[1569/73] /[1569/70]	56.6/4.4/4.3	00:15/00:22/00:39
s9234	406/400/404	[1970/508]/[1976/174] /[1972/180]	20.5/8.1/8.4	02:09/02:05/02:32
s13207	717/716/715	[2503/173]/[2504/2] /[2504/2]	6.5/0.1/0.1	01:44/01:42/01:56
s15850	278/279/280	[2368/134]/[2367/218] /[2366/235]	5.4/8.4/9.0	02:02/02:03/02:54
s35932	34/33/33	[9728/43]/[9729/93] /[9729/93]	0.5/0.9/0.9	06:40/06:31/06:45
s38417	426/426/422	[14491/11805]/[14491/2848] /[14495/1881]	44.9/16.4/11.5	28:04/34:29/46:52
s38584	257/242/243	[9467/2478]/[9482/207] /[9481/144]	20.7/2.1/1.5	11:08/10:57/13:34
chip1	1149/1151/1142	[14699/67496]/[14697/26535] /[14706/24447]	82.1/64.4/62.4	120:31/119:44/137:58
chip2a	1534/1783/1778	[5819/27772]/[5476/3618] /[5470/3569]	82.7/39.8/39.5	391:58/403:50/427:06
chip3	6288/5431/5349	[41308/2697369]/[42387/538853] /[42465/459132]	98/92.7/91.5	79hrs/29hrs/50hrs

3.5 Conclusions

We have proposed an improved dynamic compaction algorithm with recursive learning for generating compact test sets for *K* longest paths per gate (KLPG) tests. The algorithm inherits the framework of [62] and compacts paths with non-conflicting assignments together during path generation. Experimental results show that recursive learning [65] is effective in trimming the search space prior to final justification and reducing the final justification failure rate. Our results also show that our PODEM-like final justification algorithm struggles with the large number of necessary assignments in highly-compacted test patterns. Future research will explore more advanced search algorithms suitable for this problem.

4. DELAY TEST GENERATION WITH A REALISTIC LOW COST FAULT COVERAGE METRIC

4.1 Motivation

In many designs, there are a set of "speed" paths that determine the clock cycle time, and most fault sites have relatively short paths, as shown in Figure 25. The paths used in these circuits are from the *CodGen* KLPG [29] ATPG tool, which generates 2 paths (1 with longest rising transition and 1 with longest falling transition) through each line in the circuit. The upper line is the longest testable path length in each circuit, which determines the clock cycle. The lower line is the average of the longest path through each line. Clearly the average longest path per line is much shorter than the globally longest path in these circuits. For example, for s38417, the longest path is 41 gate delays, while the average is 18.1 gate delays.

Figure 26 gives more details about the path delay distribution in circuit s38417. Of the 21,855 paths, only 4.15% have nominal delay greater than 80% of t_{max} (the globally longest path length) and only 39.37% have nominal delay greater than 50% of t_{max}. This distribution is typical of many circuits. It shows that most fault sites only have short paths through them.

This section proposes a *realistic low cost fault coverage metric* targeting both global and local delay faults. It suggests the test strategy of generating a different number of longest paths for each line in the circuit while maintaining high fault coverage. This metric has been integrated into the *CodGen* ATPG tool. Experimental results show

significant reductions in test generation time and vector count on ISCAS89 and industry designs.

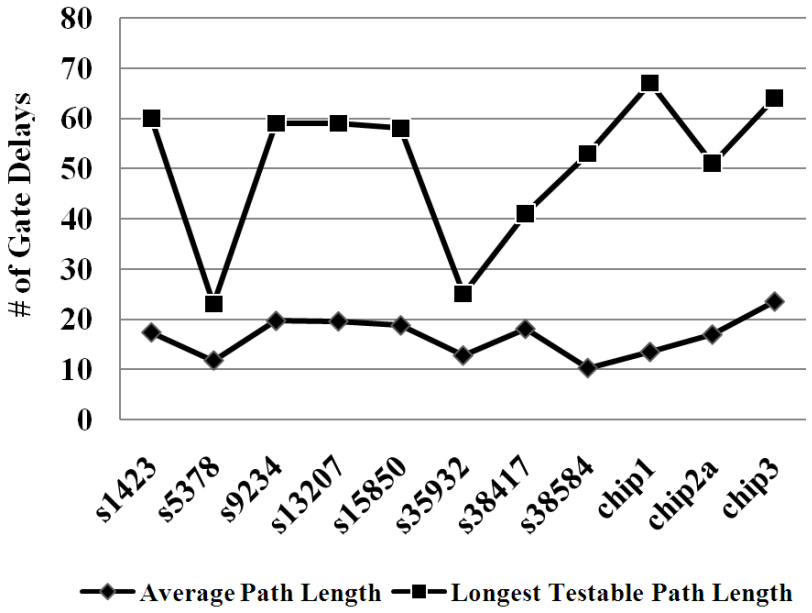


Figure 25. Path statistics for ISCAS89 circuits.

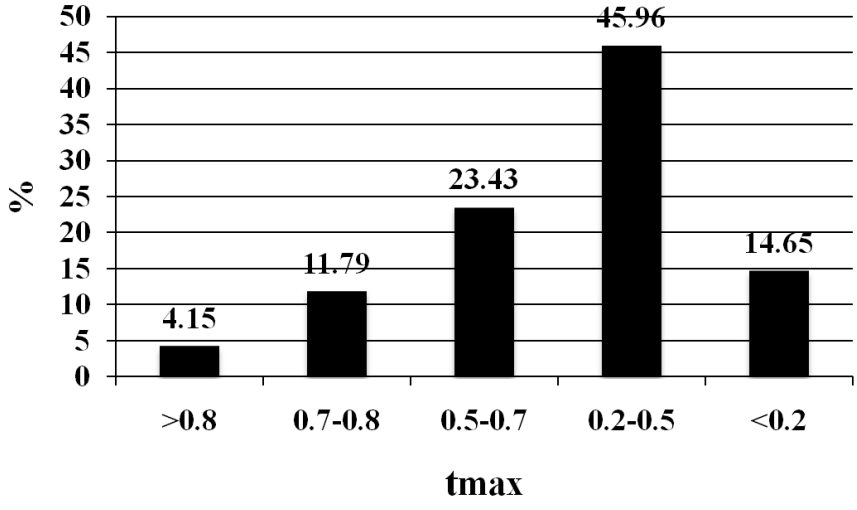


Figure 26. Path delay distribution of s38417.

4.2 Previous Work

Delay test targets small manufacturing defects to ensure that the design works well within the specified timing margin. We term delay faults caused by a local disturbance such as a resistive bridge or open *local delay faults* and those caused by global process parameter variation *global delay faults* [27]. Process variation can be quite complex [66][67]68], consisting of systematic, die-to-die random variation and intra-die random variation. The path delay fault model [17] assumes that a path has an arbitrary delay increase, so it can model the combination of local and global delay faults. However, all paths must be tested in order to achieve high fault coverage. The exponential number of paths in a circuit limits usage of this model.

Some path selection methods have been proposed to choose a subset of paths. The simplest approach is to select paths with structural delay exceeding a specified threshold, such as 90% of t_{max}, the maximum allowable circuit delay (e.g. clock cycle time or globally longest path length). However, the number of longest paths selected using min-max delay values is often too large, even with structural correlation information [69]. With manufacturing knowledge, the number of potentially longest paths can be pruned to just a few based on structural and process correlation [70]. But this method only considers the longest paths in the whole circuit, which may not cover every gate in the circuit. The approach in [71] selects longest paths for every fault site using a linear function of process-variation variables, and further considers process variation in both devices and interconnect. However, this method is difficult to integrate into test generation. In [72], a pattern selection technique uses the least slack on the scan

flip-flops to create different pattern categories. Then no-timing ATPG is performed on each category to exercise more longer paths. But this technique does not guarantee the generation of the longest paths. Another solution to this problem is to adjust the capture clock so that the timing margin on these shorter paths is reduced [73][74]. The number of different capture clock timings is reduced if the paths are divided into groups of similar length [75] or test generation is performed to achieve paths of similar length [76]. These techniques have been shown to catch delay faults that cause system failures [73]. The drawback of such faster-than-at-speed testing approaches is that they can cause overkill and test escapes. *CodGen* generates tests for the *K* Longest Paths per Gate (KLPG) or line [29], so is linear in the circuit size. Testing *K* longest paths targets global delay faults and testing every line covers local delay faults. However, the vector count is high for high *K*.

In [37], a low cost fault coverage metric combining local and global delay faults was proposed. This metric suggests the test should cover every line in the circuit by testing one of the longest paths through the line, and test more long paths through each line to increase the delay fault detection probability under process variation. At the same time, a transition fault test should be applied to detect most large delay faults. The problem with [37] is that this method can only be used to evaluate the coverage of an existing test set, not to drop faults during test generation.

4.3 Low Cost Fault Coverage Metric

Existing delay fault coverage metrics are not suitable for measuring realistic delay fault coverage. For example, a traditional path delay fault metric defines *the delay fault coverage = number of tested paths / number of total testable paths*. And some methods [77][78] have been proposed to identify untestable paths. However, it can be very expensive to calculate all testable paths in the circuit, especially for a ISCAS85 benchmark circuit c6288, with an exponential number of testable paths. Transition fault coverage is sometimes used as an index for test quality measurement, but the transition fault coverage of a circuit is the same whether short or long propagation paths are used, when long paths have higher real defect coverage.

Our existing realistic fault coverage metric [37] defines the fault coverage (FC) for test t as:

$$\mathbf{FC=P(}t\ \mathbf{detects\ delay\ fault\ |\ chip\ has\ a\ delay\ fault)} \quad (1)$$

For a given fault site i and a given extra delay Δ, the detection probability of extra delay Δ for test t can be translated from general metric expressed formula (1) into formula (2):

$$\mathbf{DP}_{i,\Delta}(t) = \mathbf{P(at\ least\ one\ tested\ path\ through}\ i\ \mathbf{is\ slow)} \quad (2)$$

Over the distribution of an arbitrary delay Δ, the detection probability for site i is calculated as:

$$\mathbf{DP}_i(t) = \int \mathbf{DP}_{i,\Delta}(t) \bullet \mathbf{p}_i(\Delta)\mathbf{d}\Delta \quad (3)$$

where $p_i(\Delta)$ is the PDF of Δ at fault site I caused by physical defects such as resistive shorts [79][80] and resistive opens [81]. And the overall fault coverage is:

$$\mathbf{FC}(t) = \sum_i \mathbf{DP}_{i,\Delta}(t) \bullet w_i \quad (4)$$

where w_i is the weight for fault site i ($\sum_i w_i = 1$).

Figure 27 [37] shows an example of this idea. There are 4 paths P_0-P_3 through a fault site, each having a delay distribution due to process variation. Assume we have a vector t only testing P_1 and the longest path P_0 is not covered. Δ_0, Δ_1 and Δ_2 are the smallest slacks for P_0, P_1, and P_2 under process variation. When $\Delta_0<\Delta<\Delta_1$, $DP_{i,\Delta}(t)$ is 0; when $\Delta>\Delta_2$, $DP_{i,\Delta}(t)$ is 100%, because the tested path P_1 is definitely slow; when $\Delta_1<\Delta<\Delta_2$, $DP_{i,\Delta}(t)$ increases from 0 to 100% as Δ increases. In order to achieve high test coverage, the longest path P_0 must be tested to eliminate the 0-DP region between Δ_0 *and* Δ_1. Potentially longest path P_2 should also be targeted to increase the DP between Δ_1 and Δ_2. The main cost to compute the fault efficiency is on the sensitization check for all the paths whose length is between Δ_1 and Δ_2, which is not easy.

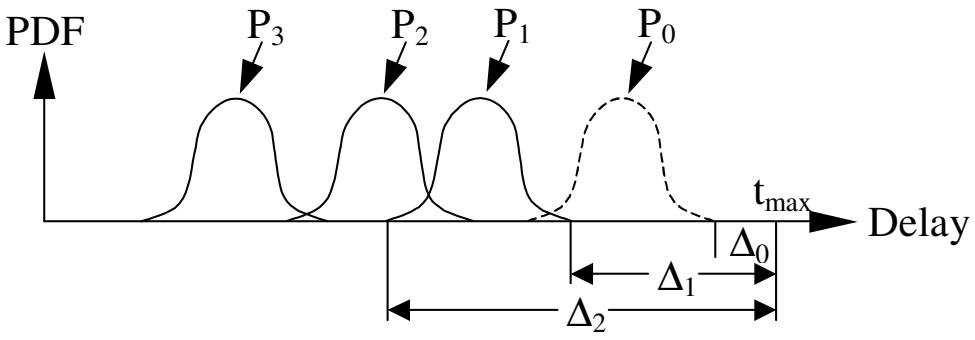


Figure 27. Fault coverage distribution.

Figure 28 shows the delay space [37] for two paths under process variation. If the paths have no correlation, the delay value combination can be any value within the rectangle. If the paths are 100% correlated, the delay value combination is a line. In reality, the correlation is somewhere between 0% and 100%, and the realistic delay space is the shaded area. Using correlation, delays on untested paths can be estimated from delays on tested paths [82], and those paths are dropped if tested ones are not faulty.

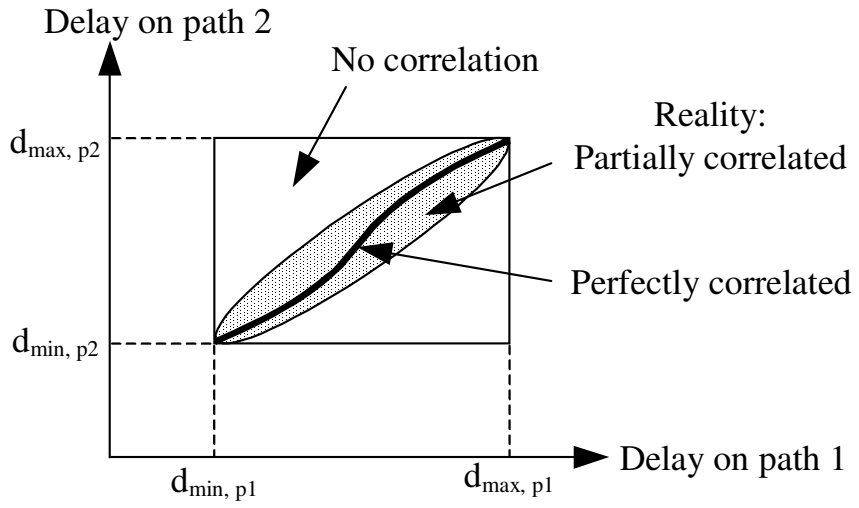


Figure 28. Delay space under different path correlations.

4.4 Realistic Low Cost Fault Coverage Metric

Our goal in using the low cost fault coverage metric is to reduce vector count by dropping faults with detection probability sufficient to achieve the desired test quality. If one path through a fault site is long and all others are much shorter, then testing the short

paths has no benefit. For example, in Figure 29, if path *A-C* has been tested, then there is no benefit in testing *A-D*, *B-C* or *B-D*, assuming these paths are never longer than *A-C* under local defects or process variation. For a fault site with several critical paths, testing more will increase detection probability. As shown in Figure 30, in addition to test path *A-C*, testing *A-D*, *B-C* and *B-D* will increase the chance of catching the defect. In the extreme case of a fault site with only short paths (Figure 31), a transition fault test is adequate and testing the longest short path will not increase defect detection probability. If a defect size is big enough to make the longest short path fail, it is very likely that this defect will make all short paths fail, since the chance that a local delay fault is big enough to make the longest short path fail, but make the little bit shorter path successful is low.

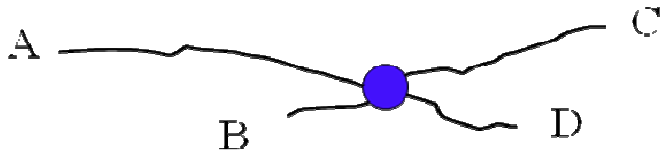


Figure 29. Fault site with short and long paths.

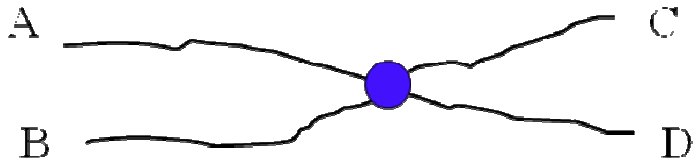


Figure 30. Fault site with long paths.

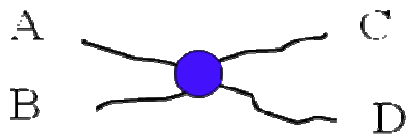


Figure 31. Fault site with short paths only.

The definition of a "short" and "long" path is based on the worst-case path delay relative to the clock cycle, process variation, and the bridge and open resistance distributions. In general, a precise physical model to reflect the real process and defect environment is not available. Even if available, it would be too costly to use during test generation. In order to minimize test generation time, a simple model is desired. In this research, we use two assumptions to simplify the problem. First, we assume the process variation is independent for each path and influences delay by increasing the required delay guard band. The percentage bound α covers the influence of inter-die and intra-die variation [83], power supply and substrate noise, and capacitive coupling. Second, we consider that the local delay defect size due to resistive short or open has a guard band. The defect size to exceed this guard band requires a bridge resistance so small or open resistance so large that it nearly causes a transition fault. Figure 32 shows the typical increase in delay vs. bridge resistance [84]. For a given fault site, when $R_{bridge} < R_F$, a resistive short behaves as a transition fault. Δ_{max} is our preset guard band value. The longest path generation will stop when the worst case path delay plus Δ_{max} is less than the specified clock cycle t_{max}. If the delay defect size is between Δ_l and Δ_{max} ($R_F<R_{bridge}<R_t$) and it causes the longest path to violate t_{max}, there will be some fault coverage loss by dropping the path, since the longest path is not considered long enough

to increase DP. If the bridge resistance is assumed uniformly distributed between 0Ω and 40kΩ [85], the possibility of fault escape is very small. Obviously the closer Δ_{max} is to Δ_1, the higher the fault coverage achieved. But it is difficult to accurately estimate Δ_1 for every fault site. For simplicity, we set Δ_{max} as several gate delays in our experiments. In practice, the more knowledge of process and defect behavior, the tighter the delay bound we can use while avoiding test escapes.

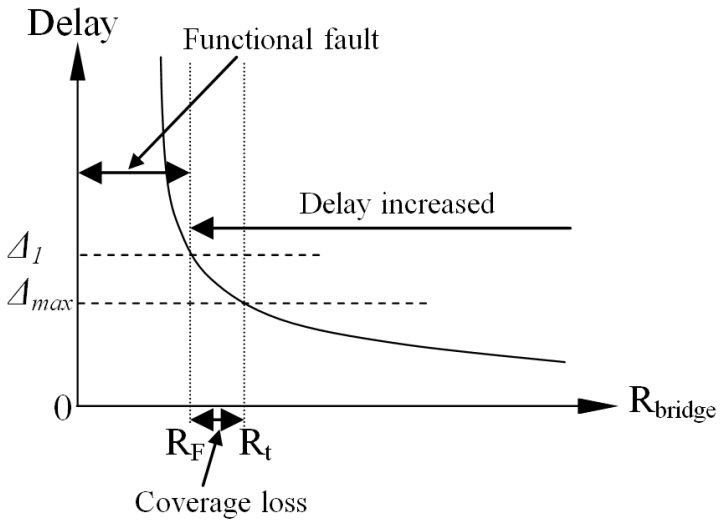


Figure 32. Example of delay vs. bridge resistance.

Based on these two assumptions, we set a detection probability threshold $P_{threshold}$ as a function of process variation (α), local delay fault guard band (Δ_{max}) and clock cycle (t_{max}), as expressed in (5):

$$P_{threshold} \bullet (1+\alpha) + \Delta_{max} = t_{max} \quad (5)$$

This threshold can be adjusted to trade test vector count vs. fault coverage.

We use a heuristic to determine whether a path can be dropped based on its nominal delay $P_{nominal}$, as expressed in (6):

$$\boldsymbol{P_{nominal} < P_{threshold}} \quad (6)$$

That is, whenever the maximum delay of a path under process variation plus local defect size guard band is less than the clock cycle time t_{max}, it is dropped from KLPG test generation. Top-off transition fault tests will be generated for those dropped fault sites that do not have any test. If a fault site has many long paths through it, the number of paths generated for this fault site (K) will be increased to increase the defect defection probability, until reaching a limit K_{max}.

4.5 KLPG with Realistic Low Cost Fault Coverage Metric

CodGen generates 2K paths (K longest with rising transition and K longest with falling transition) through each line in the circuit. It currently drops a fault site once K rising and falling paths through it have been generated. This is true even if the fault site has only short (large slack) paths passing through it. This approach is inefficient, since the probability of such a large delay due to a resistive short or open at this fault site is essentially the same as a transition fault. Similarly, if a fault site has one very long path, and the remaining paths short, and the long path has been tested, the short paths can be dropped, since they contribute nothing to the coverage (e.g. path P_3 in Figure 27). Exploiting this information will significantly reduce the vector count and ATPG time, due to an increase in fortuitous drops. We can use this savings to increase K on fault

sites with many low slack paths (such as P_2 in Figure 1) to increase DP, since paths with little timing slack provide the highest fault detection probability.

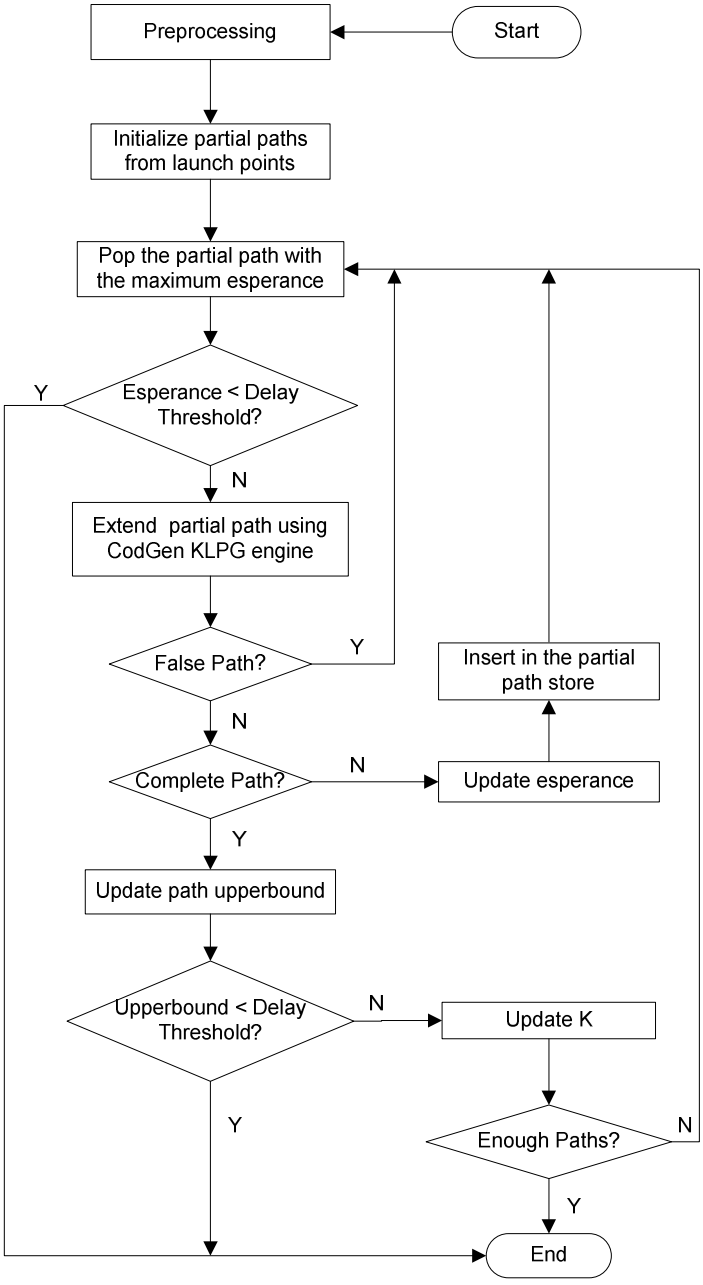


Figure 33. KLPG flow with low cost coverage metric.

Figure 33 shows the updated *CodGen* flow for a single fault site, including the proposed heuristics. In the preprocessing phase, topology information such as the *static timing analysis* (STA) delay of each gate is calculated, to help speed up the path generation. A path store is created to store partial paths, which are paths from a launch point (a primary input or scan cell output) but have not reached a capture point (a primary output or scan cell input). As introduced in section 1, e*sperance* [30] is the value associated with each partial path. It is the sum of the length of the partial path and the STA delay from its last node to a capture point. Partial paths in the path store are sorted in non-increasing order of esperance. Once the esperance of the partial path is less than $P_{threshold}$, the partial path is discarded and we stop test generation for this line. During each iteration of path generation, the first partial path in the path store is popped, and extended by adding one gate with largest max esperance. Then side input constraints to propagate the transition through the added gate under different sensitization criteria (e.g. robust or non-robust) are applied. Direct implications are used to identify local conflicts. If there is any conflict, this partial path will be identified as false and trimmed off. If the partial path becomes a complete path, a PODEM-like final justification is called to find a test vector. The path delay upper bound is updated once a complete sensitizable path is generated. Then $P_{threshold}$ will be used to determine whether to stop path generation or generate more paths through this line. If the upper bound delay is less than $P_{threshold}$, i.e. all remaining paths are too short to fail delay test, test generation for this line ends and the path store is released. On the other hand, we may increase K for lines with many

possible long paths to increase DP when the possible delay of a newly generated complete path is greater than $P_{threshold}$.

This low cost fault coverage KLPG flow works together with our dynamic compaction algorithm [62] to generate compact test sets. In the current implementation, the test generation strategy is to first generate a robust test for a line, then continue to generate non-robust test and finally a long transition test if there exists any path with delay greater than $P_{threshold}$, as long as K is less than the specified K_{max}. This approach assumes a robust path always has better test quality than a non-robust or long transition path with similar length. In our experiments, the distribution of potentially longest paths shows that most fault sites require only a few paths to be tested. For fault sites with many long paths, suppose several potentially longest paths have been tested. If the remaining potentially longest paths are only slightly longer or shorter than the tested ones, they have little fault coverage benefit, and these paths can be dropped. Thus a reasonable K_{max} will be selected to limit the vector count while maintaining high DP. A K_{max} of up to 5 was used in our experiments. The fault simulation results in [37] have indicated the fault efficiency or fault coverage saturated when increasing K value 1 to 5 for the KLPG test, as shown in Figure 34 with an example circuit c7552.

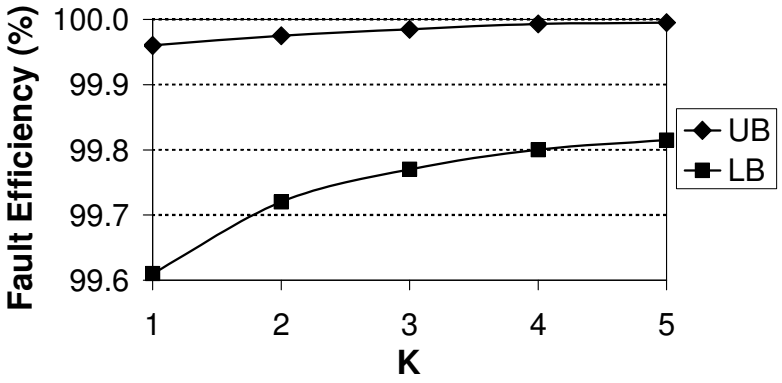


Figure 34. Fault coverage vs. K (circuit c7552).

When KLPG test generation is finished, top-off transition fault tests are generated to cover the fault sites not fortuitously detected by KLPG. These are the fault sites that can only fail in the presence of a large local delay defect in combination with process variation.

4.6 Experimental Results

We performed experiments on full scan versions of ISCAS89 benchmarks and three industry designs (chip1, chip2a and chip3). Since we only have SDF models for some circuits, we use unit gate delays for all circuits to make comparisons. The maximum delay t_{max} is set to be 8% longer than the nominal delay of the longest testable path. It is assumed that there is only one spot defect in the circuit, and the circuit is subject to process variation.

4.6.1 KLPG with Low Cost Fault Coverage Metric

In the first experiment, process variation is assumed to be ±20% of the nominal path delay and the local random spot defect guard band is 3 gate delays. We assume that local delay defects exceeding 3 gates are essentially transition faults. So $P_{threshold}$ is set to "$(t_{max}$-3)/120%"**.** Table 11 gives the results of *CodGen* using the low cost fault coverage metric in launch-on-capture test mode under different K_{max} (up to 5). Column 1 gives the circuit name. Column 2 gives the total number of lines in the circuit. Column 3 gives the number of paths tested under K_{max} values of 1, 3 and 5 respectively. K_{max}=n means the nth longest path through a line will be generated if and only if its nominal length is greater than $P_{threshold}$. Column 4 shows the dynamically compacted vector count under different K_{max}. The number of tested paths and vectors increases approximately linearly with K_{max}. For chip1 and chip2a, the number of tested paths is small, which indicates that many lines are dropped because the longest paths through them are short, and will be covered by transition fault test. The average length of the tested paths is listed in column 5. Column 6 shows the longest testable path length. As desired, the average length is close to the longest path length. Column 7 lists the transition fault coverage for the tested paths. For most circuits, the transition fault (TF) coverage is low, since the tested paths cover only a small fraction of the gates. In s35932, many paths are tested and the TF coverage is >50%. This is because this circuit is optimized to have many paths close to the maximum delay. The last column in the table reports the CPU time under different K_{max}.

Table 11. KLPG with low cost fault coverage metric (LOC with 20% process variation).

Circuit	# Paths Tested (1/3/5)	# Vectors (1/3/5)	Av Path Length (1/3/5)	Longest Testable Length	Test Coverage (%)	Time (m:s) (1/3/5)
s5378	108/286 /419	34/72/101	21.54/21.42 /21.42	23	6.04/6.17 /6.13	00:16/00:18/00:20
s9234	124/367 /611	50/97/154	49.83/49.63 /49.46	59	5.84/6.41 /6.80	06:48/06:32/06:57
s13207	8/24 /40	4/5/7	59/58.88 /58.65	59	0.82/0.82 /0.82	00:51/00:52/00:55
s15850	179/525 /847	123/353/525	56.66/56.63 /56.59	58	4.75/4.75 /4.75	03:08/03:22/03:34
s35932	5216/13984 /21648	22/68/80	22.03/21.88 /21.85	25	55.34/55.55 /55.77	11:15/31:11/43:46
s38417	1980/6057 /10029	132/240/ 405	32.43/32.43 /32.36	41	11.32/12.72 /13.28	07:58/12:13/18:50
s38584	363/1060 /1682	188/244/312	48.60/48.64 /48.66	53	2.43/2.45 /2.45	06:33/08:12/10:37
chip1	20/55/61	2/4/5	65.35/64.07 /63.66	67	0.21/0.23 /0.23	39:57/32:26/33:28
chip2a	5/15/23	4/10/14	49.8/49.8 /49.70	51	0.04/0.06 /0.09	09:48/09:28/09:43
chip3	1290/3729 /5500	384/587/692	57.26/56.80 /56.82	64	1.36/1.42 /1.44	149:50/197:37/242:13

In the second experiment, process variation is set to ±30% and the local delay defect guard band is 3 gate delays. So $P_{threshold}$ is "(t_{max}-3)/130%"**.** Table 12 gives the results. Since $P_{threshold}$ is decreased, more paths with shorter nominal length will be

generated, as shown. For example, for s38417, with K_{max}=5, 10,029 paths are tested with 20% process variation, while 18,374 paths are tested with 30% process variation. The number of test vectors is sensitive to the parameters interacting with the circuit path delay distribution. It is important to set the guard band variables to reflect the real silicon environment.

Table 12. KLPG with low cost fault coverage metric (LOC with 30% process variation).

Circuit	# Paths Tested (1/3/5)	# Vectors (1/3/5)	Av Path Length (1/3/5)	Longest Testable Length	Test Coverage (%)	Time (m:s) (1/3/5)
s5378	259/680/1017	71/147/192	19.19/19.14/19.13	23	15.12/15.26/15.19	00:17/00:22/00:25
s9234	123/348/579	67/107/158	49.50/49.80/49.55	59	6.51/7.05/7.53	08:04/07:58/08:46
s13207	986/2813/4639	484/842/1127	48.08/47.98/47.86	59	11.09/11.22/11.49	02:38/04:08/05:58
s15850	213/621/1011	131/369/549	55.31/55.30/55.15	58	5.65/5.65/5.65	03:12/03:17/03:43
s35932	5216/13984/21648	22/68/80	22.03/21.88/21.85	25	55.34/55.55/55.77	11:15/31:11/43:46
s38417	3700/11134/18374	300/530/833	32.24/32.13/32.03	41	19.85/20.22/20.55	16:14/25:05/39:15
s38584	389/1189/1811	206/289/366	48.12/47.92/48.18	53	2.56/2.67/2.63	06:44/08:49/11:33
chip1	39/95/119	10/19/29	59.97/59.31/57.92	67	0.46/0.50/0.51	32:11/33:18/34:02
chip2a	72/218/327	19/51/68	42.89/42.18/42.00	51	0.81/0.88/0.85	16:00/18:04/20:07
chip3	2580/7736/11585	751/1349/1597	53.31/52.62/52.60	64	2.98/3.18/3.22	4.8hrs/8hrs/10.2hrs

4.6.2 KLPG-5L vs. KLPG-5 vs. TF

The KLPG-5 test targets the five longest rising and five longest falling paths through each fault site. A KLPG-5 test is composed of the longest robust rising and falling paths through each line, topped off with non-robust KLPG vectors, topped off with long transition fault vectors. As shown in Figure 25, most KLPG paths are short paths. Further, KLPG test generation for fault sites covered by short paths is expensive because longer false paths must be eliminated. In addition, shorter paths do not require as many vectors as longer paths, since they have fewer necessary assignments, permitting increased test compaction. A KLPG-5L test is composed of a KLPG-5 test with the low cost fault coverage metric, which drops fault sites with only short paths, topped off with transition fault tests for those dropped fault sites. If a fault site has several potentially longest paths exceeding the guard band of the metric, KLPG-5L will continue to generate them (up to 5 in our experiments). Both KLPG-5 and KLPG-5L tests achieve the same transition fault coverage as a transition fault test set, but with higher quality, since a transition fault test has potential quality loss due to the possible propagation of glitches [8] and the uncertainty of the propagation path length. Table 13 compares the KLPG-5 test size to KLPG-5L under process variation of 20% and local delay defect guard band of 3 gate delays, both in launch-on-capture mode. Column 1 gives the circuit name. Column 2 shows the number of tested paths for the two tests. Column 3 gives the CPU time for the two tests. Column 4 shows the test generation speed up for KLPG-5L. Overall, KLPG-5L test generation is much faster than KLPG-5. The speed up is relatively small for most ISCAS89 circuits, and large for the three larger

industry circuits. For example, for chip2a, the CPU time of KLPG-5L is only 19.5 hrs compared to 100.5 hrs for KLPG-5, a 5.15x speed up. One special case is s9234, for which the speed up is 0.75x. The reason may be that top-off TF generation adds additional time. The other special case is s35932, for which the speed up is 1.05x. As explained earlier, s35932 has many near-critical paths that cannot be dropped using the low cost delay fault coverage metric.

The last three columns in Table 13 show the vector count of KLPG-5, KLPG-5L and Transition Fault (TF) test. The TF test was generated by a commercial tool. The KLPG-5 test size is high for some large circuits, such as chip3 (19863 vectors). With the implementation of the low cost coverage metric, the KLPG-5L test has a much smaller test size (1296 vectors) with reasonable CPU time. Since top-off TF vectors are generated for the KLPG-5L test, the KLPG-5L test has the same TF coverage as KLPG-5 and TF. For all circuits, our combined test set is smaller or only modestly larger than the transition fault test set, but with higher quality.

Table 13. KLPG-5 vs. KLPG-5L.

Circuit	# Paths Tested (KLPG-5/5L)	Time (m:s) (KLPG-5/5L)	Speedup Factor	# Vectors		
				KLPG-5	KLPG-5L	TF
s5378	8125/2779	02:00/01:00	2.00	559	253	194
s9234	14543/3984	10:06/13:27	0.75	957	230	465
s13207	21375/5974	13:16/07:16	1.83	2248	358	382
s15850	19132/5315	12:50/08:02	1.60	1169	612	231
s35932	56369/27554	34:57/33:22	1.05	86	72	68
s38417	90739/25395	145:34/63:24	2.30	1415	255	365
s38584	63082/19673	102:07/39:34	2.58	826	514	528
chip1	107782/31947	548:00/341:19	1.61	3388	1602	1900
chip2a	99052/13492	100.5hrs/19.5hrs	5.15	9464	1199	2537
chip3	186733*/82373	146hrs*/64.8hrs	2.25*	19863*	1296	1445

*For chip3, only robust tests are generated in the KLPG-5 test.

4.7 Conclusions

We have developed a realistic low cost fault coverage metric that considers both process variation and local delay fault. Simple heuristics based on process variation and local delay defect sizes are used to filter out paths and accelerate *CodGen* KLPG generation [86]. Transition fault tests are generated for dropped fault sites to ensure the test quality. Experimental results show that path generation with this fault coverage metric is efficient and the vector count is practical. Monte Carlo simulation and real silicon data will be needed to verify the effectiveness of this method.

During test generation, we currently treat path delays as independent. This is increasingly realistic for gate delays, due to random dopant fluctuation, but is not true for interconnect delays. In the future, structural and spatial correlation will be explored to reduce the guard band. This can be particularly beneficial for circuits with many near-critical paths.

5. EXPERIMENTS ON SILICON

The industrial design selected for the experiments is an AMD quad-core microprocessor. The microprocessor is a 45 nm technology design and contains a total of 11.2M gates in the test model, excluding embedded arrays and scan flip-flops. Each core test model contains 3.4M gates, 160,000 LSSD-type scan flip-flops, and 746,000 unscanned flip-flops (that are contained in the surrounding untested cores and logic). There are several clock domains, but ATPG only runs for a given clock domain using launch-on-capture.

5.1 Flow of AMD Experiments

The complexity of a microprocessor brings new challenges for KLPG test. A set of tools have been developed in our lab for the AMD experiments. Figure 35 shows the complete KLPG test generation flow. The right side lists all the necessary files for each stage. In the first step, the Verilog netlist and library files are fed into a parser. The parser will flatten the hierarchical netlist and library and parse the hierarchical names into an easy-to-parse format used by *CodGen*. Then the hierarchical information is not available anymore in the flattened design. Each cell/net is renamed with simple indices, such as U100, N150, etc. The original library file had to be modified, since some cells were described using Verilog data flow constructs, while the parser only supports structural Verilog.

In the second step, a scan chain tracing tool traces the scan chain forward from scan input to scan output to identify every scan cell in the flattened netlist, and the logic

between the scan cells. The parsed scan chain report will then be verified against the Mentor Graphics FastScan scan chain report.

In the third step, the FastSscan do file and procedure file are used to map the test constraints and clocks in the parsed netlist. The do file and procedure file specify test pin and internal preset values, clock sources, test clocking scheme, functionality of each test pin, etc. The files for transition fault test are reused for KLPG test. The constraints and clocks must be mapped correctly, or the test will be invalidated, and potentially the chip will be damaged.

In the test generation stage, $2K$ longest paths through each line (K paths with rising transitions and K paths with falling transitions) are generated. In these experiments, K=1. Statically-compacted patterns are saved in ASCII format for later processing. The steps prior to test generation only need to be run once per design, as long as the design is not modified.

The ASCII-format test patterns cannot be applied directly on the tester. Any mismatches in scan-out data must be identified and masked by FastScan re-simulation, which is a standard step used in industry. Miscompares on target paths to good machine values and outputs on all untargeted side paths will be masked. Then standard test interface language (STIL) format patterns are written out by FastScan for tester use.

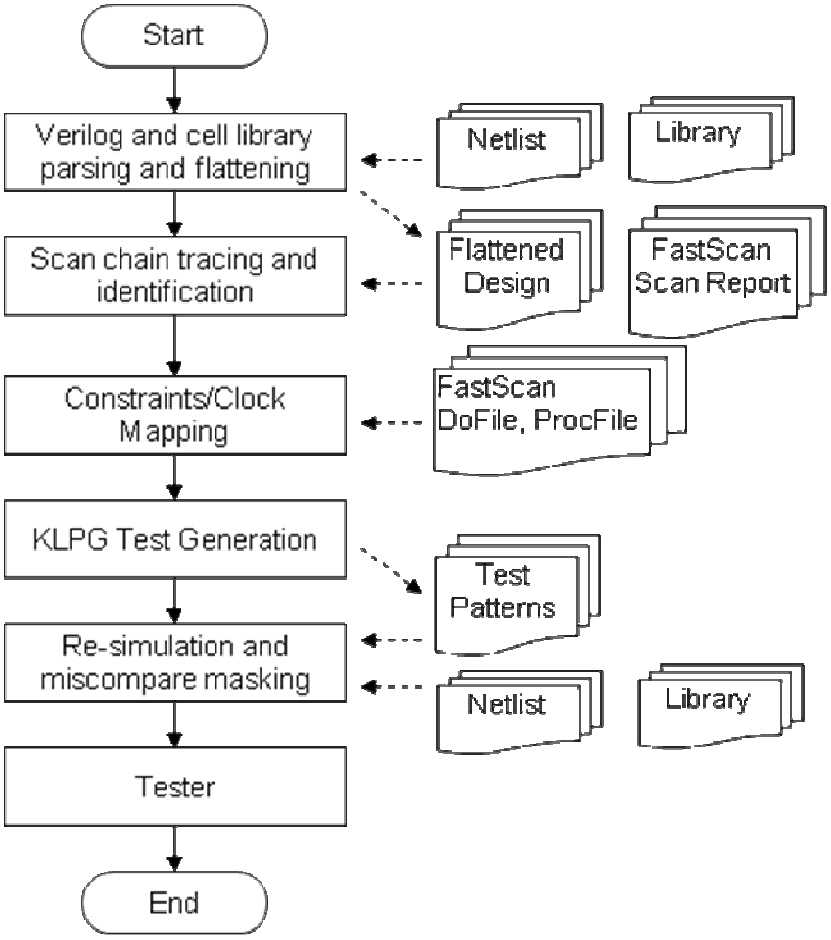


Figure 35. KLPG flow for AMD design.

5.2 Improved KLPG for AMD Design

A number of improvements were made to *CodGen* to handle the AMD processor. First, LSSD-type scan cells had to be supported, and then support for extensive and complex clock gating had to be added. Power dissipation is a critical factor in microprocessors. Clock power usually consumes a 30-35% of total microprocessor power [87], due to power consumed by combinational logic fed by the clock signals,

flip-flops and clock buffer tree in the design. Clock gating is one technique used on many synchronous circuits for power-saving. The basic idea is to turn off the clock when it is not needed. To save power, additional logic is added into the clock tree to disable portions of the circuitry. For flip-flops in a clock gated-off part of the circuit, their status is unchanged, so that their switching power consumption is zero.

Extensive clock gating is implemented in AMD microprocessors to balance the power of the whole chip. There are two types of clock gaters in this design, coarse clock gater and fine clock gater. Coarse clock gaters that control large regions of the chip are under direct scan chain control, so do not require additional ATPG effort to justify their values. Fine clock gaters must be justified at the launch flip-flop on the launch cycle, and justified at the capture flip-flop on the capture cycle. Our approach is to find the path first, and then justify the clocks for the complete path. In the AMD circuits, most clocks can be justified within the standard two capture cycles of a launch-on-capture test. To simply the clock path justification, *CodGen* only justifies the clock on the target path. For the clocks on all side paths, ATPG assumes they are justified. This leads to miscompares in pattern simulation for the side path outputs. These miscompares are masked during the FastScan re-simulation step.

Figure 36 shows an example of clock path justification. Clouds stand for the combinational logic. *G1* and *G2* comprise a standard LSSD scan cell. If there is a '1' assigned to *Q* output of *G1* in the second clock cycle, ATPG will try to assign and justify '1' on *D* input of *G1* in the first clock cycle in launch-on-capture mode, since the final value on *Q* is derived from the initial value on *D*. Additionally, clock pin *C* of *G1* must

be justified to be active to capture the value from the *D* pin to the *Q* pin. In the clock tree feeding pin *C* of *G1*, a 2-input AND gate *A1* is fed by a clock source and the output of a 3-input AND gate *A2*. *A2* is fed by a coarse clock gater (*G5*) and two fine clock gaters (*G3* and *G4*). *G3* and *G4* are L1 latches of two LSSD cells. The value of *G5* is preset to 1 by ATPG constraints. In order to make the clock active on the *C* pin of *G1* in the capture cycle, the initial value on the output of *A2* must be '1' to activate the clock on *C*. When ATPG continues backtracing *A2* to its three inputs, two fine clock gaters *G3* and *G4* must be '0' and '1' respectively in the first clock cycle. The test is only valid when there is no conflict on *G3* and *G4*.

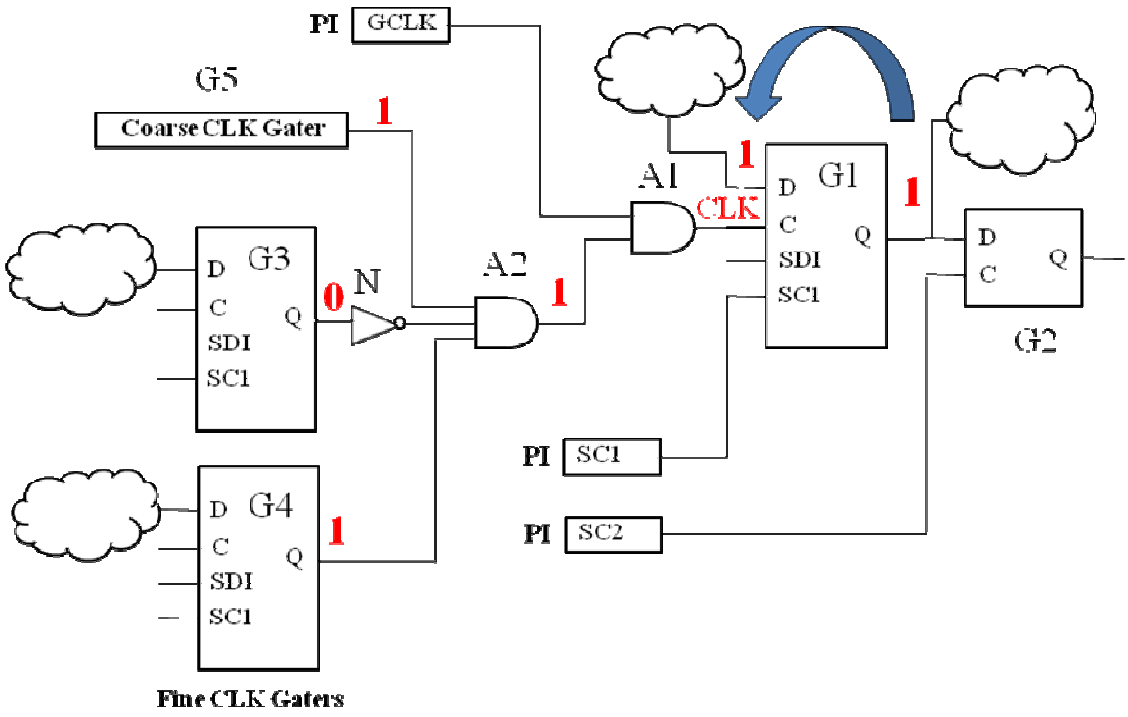


Figure 36. Example of clock path justification.

5.3 Experimental Results

Experiments are performed on two small modules in the microprocessor chip and then extended to one whole core.

5.3.1 KLPG Results for Two Modules

Table 14 shows the ATPG results of KLPG-1 test for module 1 and module 2. Module 1 has 200K gates and relatively few unscanned flip-flops, and is relatively easy to test for both TF and KLPG tests. It does not contain embedded memory. Note that the vast majority of paths have a robust test or non-robust test, so the overall KLPG test quality is very high. Module 2 contains 214K gates and more unscanned FFs, and features such as half-cycle paths that *CodGen* currently does not support. This makes it difficult for KLPG test, in coverage, CPU time and pattern count. The test quality is still high in terms of path robustness. Commercial transition fault test generates few test patterns, but the coverage is still poor.

Table 14. KLPG-1 test for module 1 & module 2.

Circuit	Module 1	Module 2
# KLPG-1 Patterns	1229	6860
# KLPG-1 Paths	208252	60643
# Robust Paths	205260	55151
# Non-robust Paths	2452	5202
# Long TF Paths	540	290
TF Coverage (%)	94.5	76.7

5.3.2 Experimental Results for Microprocessor Core

One microprocessor core test model is comprised of 3.4M gates, 160,000 LSSD-type scan flip-flops, and 746,000 unscanned flip-flops (that are in the rest of the chip). Among the unscanned flip-flops, 636,000 have uncontrollable values in them, while the remainder are constant 0 or 1, or transparent, after test initialization.

A. KLPG for Microprocessor Core

Since four cores in this quad-core microprocessor are identical, we only generated robust KLPG tests with K=1 for one core. The test patterns for one core can be later applied to all cores in test mode. Table 15 shows the ATPG results. To fit ATPG into our test schedule, dynamic compaction was not performed. The low cost fault coverage metric was not used, due to lack of information about the process. Test generation was performed on an eight-core processor, with the ten separate ATPG processes taking in aggregate 92.5 CPU days, using approximately 10 GB memory per process. The statically compacted KLPG test with one longest robust rising and falling path per line has 143,094 patterns testing 1,239,752 distinct paths. The care bit density of KLPG patterns is 0.35%, with a max density of 5.42% on the first pattern. The transition fault test for this core has 21,600 patterns. KLPG has a significantly larger test volume than TF test. This is partly due to the fact that KLPG test only uses static compaction [29][88][89] while the transition fault tests use advanced dynamic compaction algorithms [90][91][92]. The other reason is that the transition fault tests utilize test compression techniques [93][94][95][96][97] in microprocessor to further

reduce the test size. Test compression can normally provide 10X to 100X or even larger reduction in test data volume. Test compression methodologies are not supported by *CodGen*. Dynamic compaction should reduce the pattern count by 4x.

Table 15. Robust KLPG results for AMD microprocessor.

# Patterns	143,094
# Paths	1,239,752
CPU Time	92.5 Days

B. FMAX Test

FMAX is defined as the maximum clock frequency at which the circuit can function correctly for all the test patterns, at a specified power supply voltage. FMAX tests were performed on 100 chips with 143,094 KLPG test patterns and with 21,600 high compressed transition fault (TF) test patterns respectively.

FMAX test was applied to all four cores in 100 chips that pass system test. For all 400 cores in 100 chips, experimental results show a distribution of FMAX across the chips for KLPG.

Figure 37 shows the FMAX distribution for KLPG test for Core0 of 100 chips. Chips with minimal FMAX are used as reference. Three chips (chip 67, 96 and 97) with minimal FMAX are slower parts. For other chips, KLPG FMAX is between 7.3MHz and 330MHz higher.

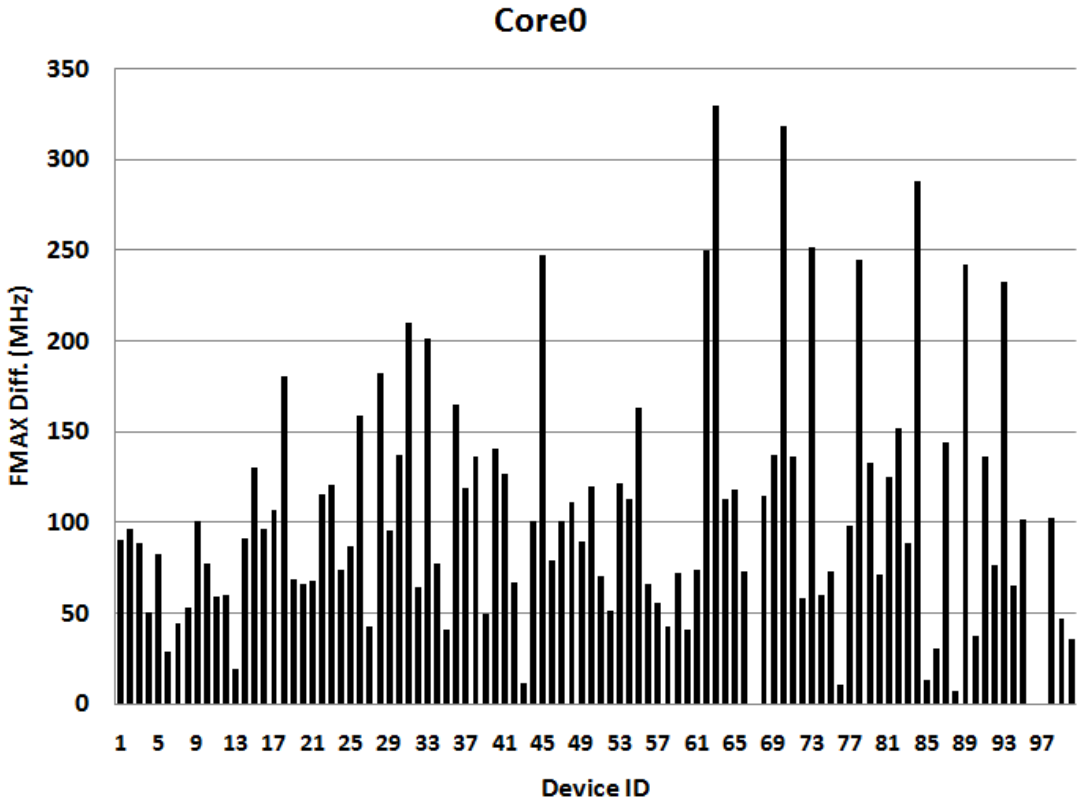


Figure 37. FMAX distribution for robust KLPG test (Core0).

Figure 38 shows the FMAX distribution for KLPG test for Core1 of 100 chips.

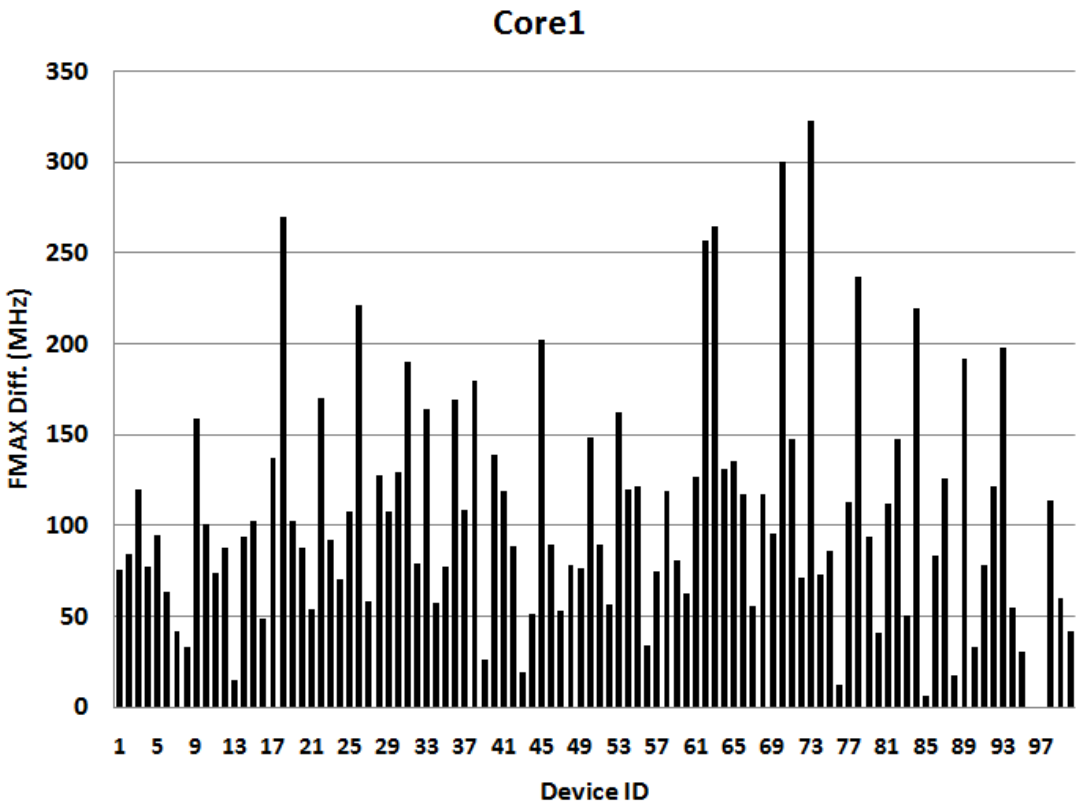


Figure 38. FMAX distribution for robust KLPG test (Core1).

Figure 39 shows the FMAX distribution for KLPG test for Core2 of 100 chips.

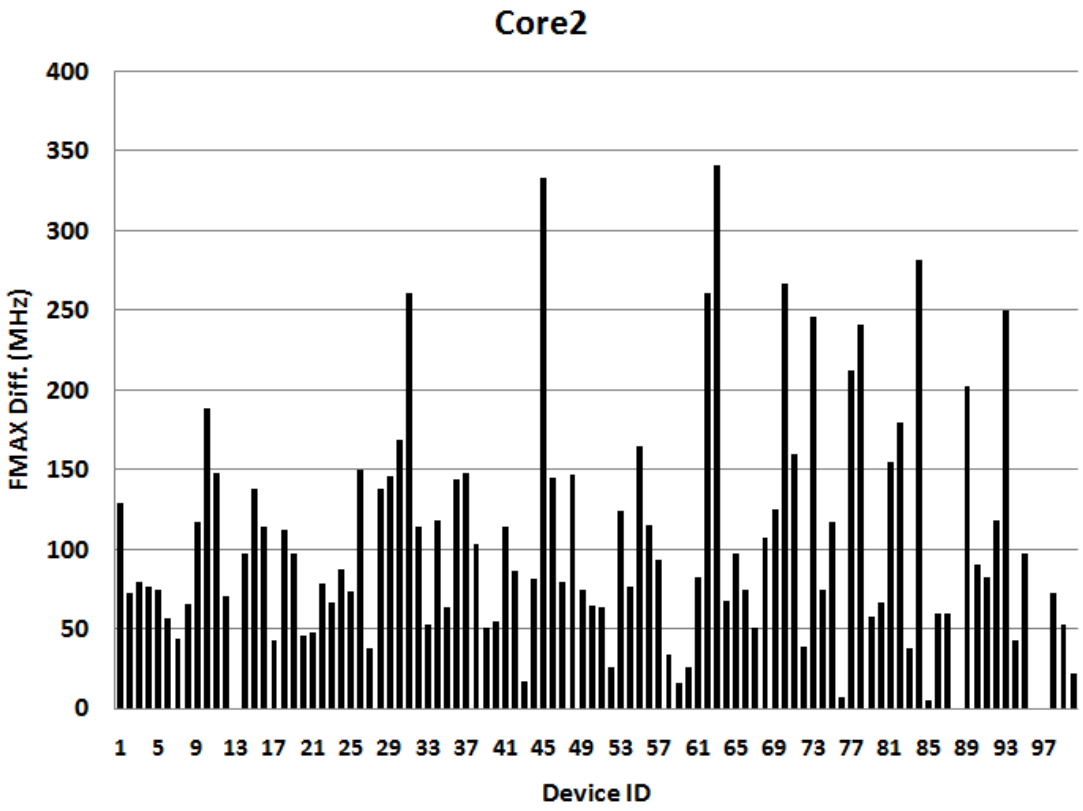


Figure 39. FMAX distribution for robust KLPG test (Core2).

Figure 40 shows the FMAX distribution for KLPG test for Core3 of 100 chips.

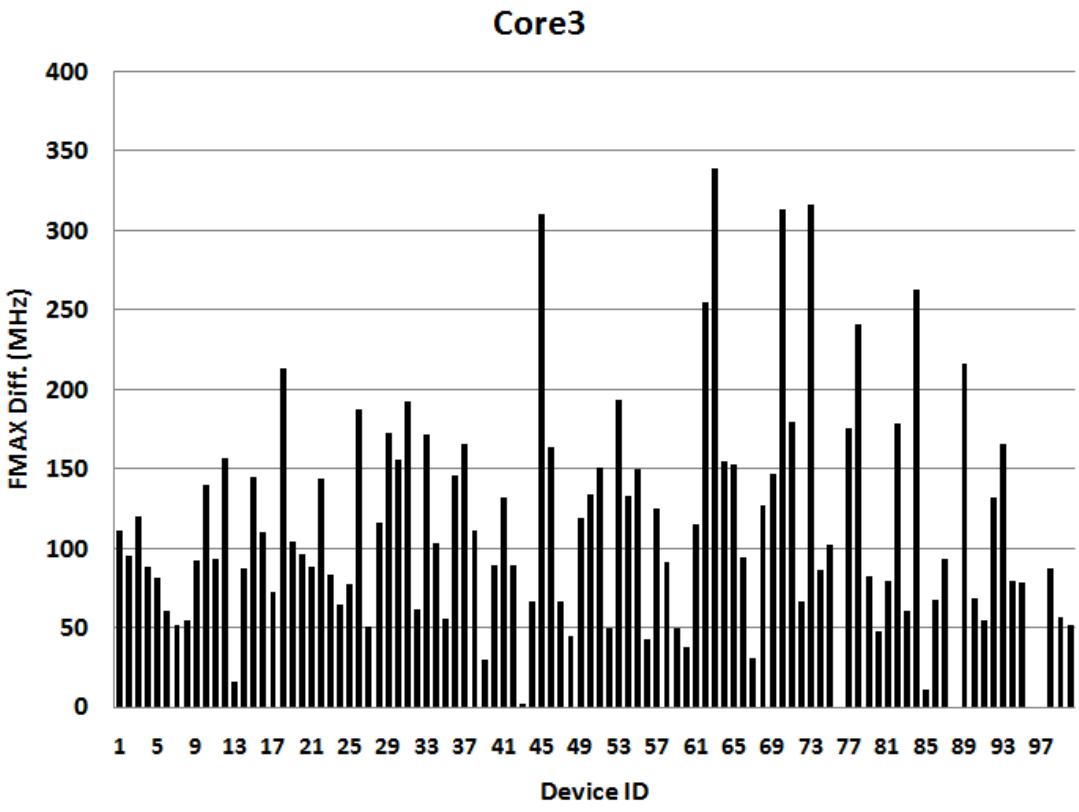


Figure 40. FMAX distribution for robust KLPG test (Core3).

Figure 41 shows the average FMAX distribution for KLPG test in chip level.

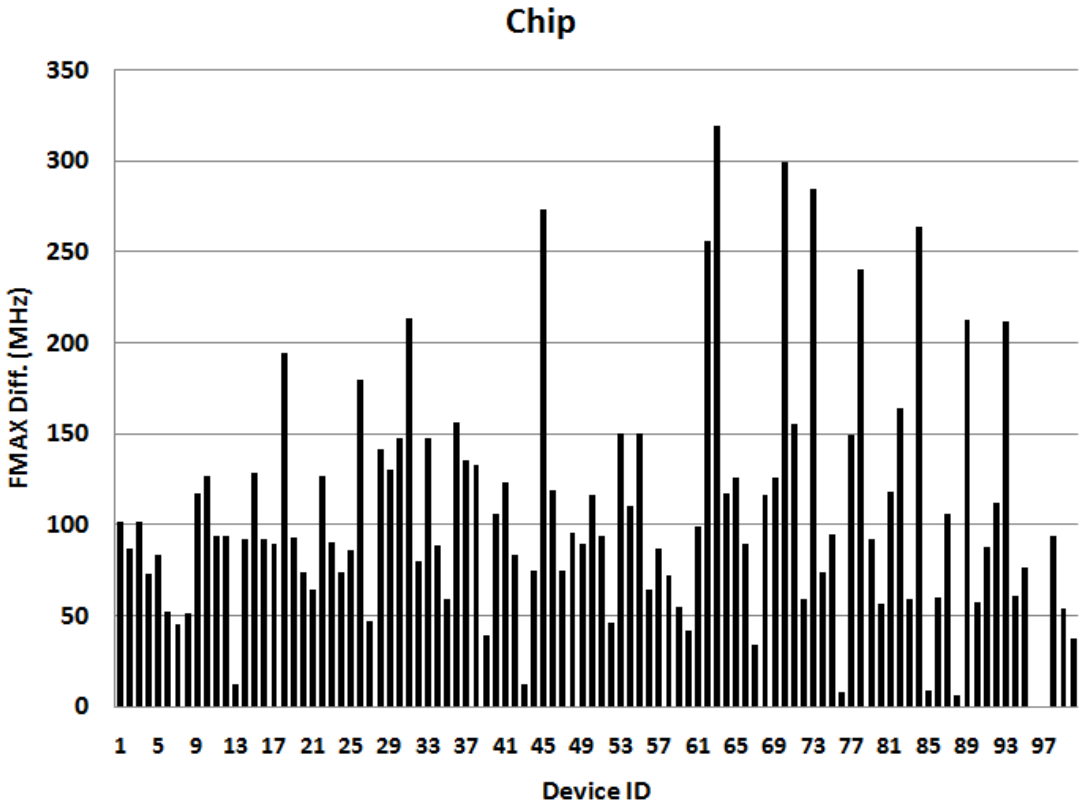


Figure 41. FMAX distribution for robust KLPG test (chip level).

In summary, Figures 37-41 give the FMAX distribution of KLPG test. It indicates some chips are faster than others. Since only robust paths are targeted in the current experiments, the small defects can only be tested through non-robust test or long TF paths are missed, which introduces some coverage loss. To further improve the test quality of our test, top-off non-robust and long transition fault paths must be added. However, if test volume and ATPG time is a concern, the transition fault test can be the top-off test for robust KLPG test.

6. SUMMARY AND FUTURE WORK

6.1 Summary

This dissertation focused on reducing the pattern count and increasing the test quality for small delay defect test. To reduce the test size for small delay test, a new dynamic compaction algorithm for generating compacted test sets for *K* longest paths per gate (KLPG) in combinational circuits or scan-based sequential circuits has been developed. This algorithm uses a greedy approach to compact paths with non-conflicting assignments together during test generation. To make our dynamic compaction approach practical for industrial use, recursive learning algorithm has been implemented to identify more necessary assignments for each path, so that the path to test pattern matching using necessary assignments is more accurate. Experimental results for ISCAS89 benchmark circuits and industry circuits show that the pattern count of KLPG can be significantly reduced (up to 4x compared to static compaction) using the proposed method. The pattern count after dynamic compaction is comparable to the number of transition fault tests, while achieving higher test quality. This algorithm is generic in nature and can be applied to any test generation procedure.

To increase the test quality, a realistic low cost fault coverage metric targeting both global and local delay faults has been developed. This metric considers the combined effects of spot defects and process variation, and takes advantage of inter-die process correlation so that the coverage is much closer to the real test quality. It suggests the test strategy of generating a different number of the longest paths for each line in the

circuit while maintaining high fault coverage. Using our low cost fault coverage metric, the number of paths and type of test depends on the timing slack of the paths. For those fault sites that do not have any robust or nonrobust tests, transition fault tests are generated for the longest paths through those sites. This metric has been implemented into the *CodGen* ATPG tool. Experimental results show significant reductions in test generation time and vector count on ISCAS89 and industry designs.

A complete test generation flow has been successfully implemented on an AMD quad-core microprocessor. *CodGen* has been improved to deal with the complex clock gating and LSSD-type scan cells. Silicon data has been collected to show the advantages of KLPG test. *CodGen* is the first university ATPG tool capable of generating small delay defect test for a commercial microprocessor.

6.2 Future Work

There are several areas that future work can focus on, as discussed below.

6.2.1 Advanced Search Algorithm

The *CodGen* KLPG ATPG uses a PODEM-like algorithm for final justification once a complete path has been identified, free of direct implication conflicts. For individual paths, this algorithm had a high success rate and relatively low CPU cost. However, for larger circuits (1M+ gates), and in dynamic compaction, this algorithm is too expensive, taking substantially more than 50% of the total CPU time. As is the case for dynamic compaction, we will consider a new search algorithm suitable for our path

delay test problem. The current KLPG algorithm uses about 10 GB of memory to generate robust KLPG paths for a 3.4M gate processor core (excluding flip-flops), or almost 3 KB/gate. This does not include any in-memory pattern storage or auxiliary data structures for dynamic compaction, which roughly doubles memory consumption. This memory consumption must be drastically reduced to make the KLPG algorithm practical in industry. The development of a new search algorithm will include attention to memory efficiency as well.

6.2.2 Pseudo-Functional Test

Pseudo-functional test [98][99][100][101] constrains the test patterns to be functional states or near-functional states. If the pattern is a functional state, then these tests can be viewed as short bursts of functional tests. The challenge in pseudo-functional test is that it is a form of sequential test. A straightforward extension of our KLPG test approach would be to use time frame expansion to search for and justify the *K* longest sensitizable path over multiple cycles through each line in the circuit. This would implicitly handle test of time borrowing schemes or setup/hold time metastability issues.

6.2.3 Test Generation for Signal Crosstalk

Delay increase or decrease due to capacitive coupling is considered in the *CodSim* delay fault simulator [102], but is not considered in the *CodGen* ATPG. If a capacitively coupled line has a transition that is the opposite of the target path, and the transition timing is aligned with the target path transition, it will cause the path to slow

down. Alignment is uncertain due to process variation, supply noise, crosstalk and delay defect. This can be viewed as an integer optimization problem, since the goal is to generate the largest delay increase by sensitizing the set of coupled transitions that causes the largest delay increase on the target path. A more efficient way to deal with coupling faults in *CodGen* will be explored in the future.

6.2.4 Dynamic Compaction Considering Power Supply Noise and Power Dissipation

Another challenge in dynamic compaction is incrementally estimating power supply noise [103] and power dissipation during the compaction process, and the corresponding path delay changes for all paths currently compacted into the pattern. The future test generation should keep noise and power at a reasonable level. An initial supply noise-aware dynamic compaction framework has been proposed in [104] to control power supply noise during test generation.

REFERENCES

[1] R. D. Eldred, “Test Routines Based on Symbolic Logical Statements," *Journal of the ACM*, vol. 6, pp. 33-36, 1959.

[2] J. A. Waicukauski, E. Lindbloom, B. K. Rosen, and V. S. Iyengar, “Transition Fault Simulation," *IEEE Design and Test of Computers*, vol. 4, no. 2, pp. 32-38, 1987.

[3] K. -T. Cheng, “Transition Fault Testing for Sequential Circuits,” *IEEE Transactions on Computer Aided Design of Integrated Circuits and System*, vol. 12, no.12, pp. 1971-1983, Dec. 1993.

[4] Semiconductor Industries Association, *International Technology Roadmap for Semiconductors (ITRS)*, Munich, Germany, 2005.

[5] Z. Barzilai and B. K. Rosen, “Comparison of AC Self-Testing Procedures,” in *Proceedings of IEEE International Test Conference*, pp. 89-94, Oct. 1983.

[6] E. S. Park, M. R. Mercer, and T. W. Williams, “Statistical Delay Fault Coverage and Defect Level for Delay Faults,” in *Proceedings of IEEE International Test Conference*, pp. 492-499, Sept. 1988.

[7] C. W. Tseng and E. J. McCluskey, “Multiple-Output Propagation Transition Fault Test,” in *Proceedings of IEEE International Test Conference*, pp. 358-366, Oct. 2001.

[8] X. Lin and J. Rajski, “Propagation Delay Fault: A New Model to Test Delay Faults,” in *Proceedings of IEEE Asian South Pacific Design Automation Conference*, pp. 178-183, 2005.

[9] J. Carter, V. Iyengar, and B. Rosen, "Efficient Test Coverage Determination for Delay Faults," in *Proceedings of IEEE International Test Conference*, pp. 418-427, Sept. 1987.

[10] V. S. Iyengar, B. K. Rosen, and I. Spillinger, "Delay Test Generation 1 – Concepts and Coverage Metrics," in *Proceedings of IEEE International Test Conference*, pp. 857-866, Sept. 1988.

[11] V. S. Iyengar, B. K. Rosen, and I. Spillinger, "Delay Test Generation 2 – Algebra and Algorithms," in *Proceedings of IEEE International Test Conference*, pp. 867-876, Sept. 1988.

[12] A. K. Pramanick and S. M. Reddy, "On the Fault Coverage of Gate Delay Fault Detecting Tests," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 1, pp. 78-94, Jan. 1997.

[13] A. K. Majhi, J. Jacob, L. M. Patnaik, and V. D. Agrawal, "On Test Coverage of Path Delay Faults," in *Proceedings of International Conference on VLSI Design*, pp. 418-421, Jan. 1996.

[14] A. K. Majhi, V. D. Agrawal, J. Jacob, and L. M. Patnaik, "Line Coverage of Path Delay Faults," *IEEE Transactions on VLSI Systems*, vol. 8, no. 5, pp. 610-613, Oct. 2000.

[15] A. K. Majhi and V. D. Agrawal, "Tutorial: Delay Fault Models and Coverage," in *Proceedings of International Conference on VLSI Design*, pp. 364-369, Jan. 1998.

[16] S. R. Nassif, "Modeling and Analysis of Manufacturing Variations," in *Proceedings of IEEE Custom Integrated Circuits Conference*, pp. 223-228, May 2001.

[17] G. L. Smith, "Model for Delay Faults Based upon Paths," in *Proceedings of IEEE International Test Conference*, pp. 342-349, Oct. 1985.

[18] W. Qiu and D. M. H. Walker, "Testing the Path Delay Faults for ISCAS85 Circuit c6288," in *Proceedings of IEEE International Workshop on Microprocessor Test and Verification*, pp. 38-43, May 2003.

[19] E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for LSI Testability," in *Proceedings of ACM/IEEE Design Automation Conference*, pp. 462-468, June 1977.

[20] F. Motika, N. Tendolkar, C. Beh, W. Heller, C.Radke, and P. Nigh, "A Logic Chip Delay-test Method Based on System Timing," *IBM Journal of Research and Development*, vol. 34, no.2/3, pp. 299-312, March/May 1990.

[21] S. DasGupta, P. Goel, R. G. Walther, and T. W. Williams, "A Variation of LSSD and Its Implications on Design and Test Pattern Generation in VLSI," in *Proceedings of IEEE International Test Conference*, pp. 63-66, Nov. 1982.

[22] C. T. Glover and M. R. Mercer, "A Method of Delay Fault Test Generation," in *Proceedings of ACM/IEEE Design Automation Conference*, pp. 90-95, Jun. 1988.

[23] B. I. Dervisoglu and G. E. Strong, "Design for Testability: Using Scan Path Techniques for Path-delay Test and Measurement," in *Proceedings of IEEE International Test Conference*, pp. 365-374, Oct. 1991.

[24] J. Savir, "Skewed-Load Transition Test: Part I, Calculus," in *Proceedings of IEEE International Test Conference*, pp. 705-713, Sept. 1992.

[25] S. Patel and J. Savir, "Skewed-Load Transition Test: Part II, Coverage," in *Proceedings of IEEE International Test Conference*, pp. 714-722, Sept. 1992.

[26] J. Savir and S. Patel, "Broad-Side Delay Test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 8, pp. 1057-1064, Aug. 1994.

[27] D. M. H. Walker, "Tolerance of Delay Faults," in *Proceedings of IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, pp. 207-216, 1992.

[28] W. Qiu and D. M. H. Walker, "An Efficient Algorithm for Finding the K Longest Testable Paths Through Each Gate in a Combinational Circuit," in *Proceedings of IEEE International Test Conference*, pp. 592-601, Sept. 2003.

[29] W. Qiu, J. Wang, D. M. H. Walker, D. Reddy, X. Lu, Z. Li, W. Shi, and H. Balachandran, "K Longest Paths Per Gate (KLPG) Test Generation for Scan-Based Sequential Circuits," in *Proceedings of IEEE International Test Conference*, pp. 223-231, Oct. 2004.

[30] J. Benkoski, E. V. Meersch, L. J. M. Claesen, and H. D. Man, "Timing Verification Using Statically Sensitizable Paths," *IEEE Transactions on Computer-Aided Design*, vol. 9, no. 10, pp. 1073-1084, Oct. 1990.

[31] A. Krstic and K. -T. Cheng, *Delay Fault Testing for VLSI Circuits.* Springer, New York, 1998.

[32] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Transactions on Computers*, vol. C-30, no. 3, pp. 215-222, Mar. 1981.

[33] Y. Shao, S. M. Reddy, I. Pomeranz, and S. Kajihara, "On Selecting Testable Paths in Scan Designs," in *Proceedings of IEEE European Test Workshop*, pp. 53-58, May 2002.

[34] W. N. Li, S. M. Reddy, and S. K. Sahni, "On Path Selection in Combinational Logic Circuits," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 1, pp. 56-63, Jan. 1989.

[35] A. Murakami, S. Kajihara, T. Sasao, I. Pomeranz, and S. M. Reddy, "Selection of Potentially Testable Path Delay Faults for Test Generation," in *Proceedings of IEEE International Test Conference*, pp. 376-384, Oct. 2000.

[36] M. Sharma and J. H. Patel, "Finding a Small Set of Longest Testable Paths That Cover Every Gate," in *Proceedings of IEEE International Test Conference*, pp. 974-982, Oct. 2002.

[37] W. Qiu, X. Lu, J. Wang, Z. Li, D. M. H. Walker, and W. Shi, "A Statistical Fault Coverage Metric for Realistic Path Delay Faults," in *Proceedings of IEEE VLSI Test Symposium*, pp. 37-42, Apr. 2004.

[38] W. Qiu, D. M. H. Walker, N. Simpson, D. Reddy, and A. Moore, "Comparison of Delay Tests on Silicon," in *Proceedings of IEEE International Test Conference*, pp. 1-10, Oct. 2006.

[39] I. Hamzaoglu and J. H. Patel, "Compact Two-pattern Test Set Generation for Combinational and Full Scan Circuits," in *Proceedings of IEEE International Test Conference*, pp. 944-953, Oct. 1998.

[40] T. M. Niermann, R. K. Roy, J. H. Patel, and J. A. Abraham, "Test Compaction for Sequential Circuits," *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 2, pp. 260-267, Feb. 1992.

[41] K. O. Boateng, H. Konishi, and T. Nakata, "A Method of Static Compaction of Test Stimuli," in *Proceedings of IEEE Asian Test Symposium*, pp. 137-142, Nov, 2001.

[42] P. Goel and B. C. Rosales, "Test Generation and Dynamic Compaction of Tests," in *Digest of Papers 1979 IEEE International Test Conference*, pp. 189-192, 1979.

[43] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System," *IEEE Transactions on Computer-Aided Design*, vol. 7, no. 1, pp. 30-35, Jan. 1988.

[44] L. N. Reddy, I. Pomeranz, and S. M. Reddy, "ROTCO: A Reverse Order Test Compaction Technique," in *Proceedings of IEEE Euro-ASIC Conference*, pp. 189-194, Jun. 1992.

[45] I. Pomeranz and S. M. Reddy, "On Static Compaction of Test Sequences for Synchronous Sequential Circuits," in *Proceedings of ACM/IEEE Design Automation Conference*, pp. 215-220, Jun. 1996.

[46] G. Ruifeng, I. Pomeranz, and S. M. Reddy, "On Improving Static Test Compaction for Sequential Circuits," in Proceedings of IEEE International Conference on VLSI Design, pp. 111-116, Jan. 2001.

[47] J. Wang, "Power Supply Noise in Delay Testing," Ph.D. dissertation, Texas A&M University, College Station, TX, 2007.

[48] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (2nd edition).* The MIT Press, Cambridge, MA, 2001.

[49] J. S. Chang and C. S. Lin, "Test Set Compaction for Combinational Circuits," *IEEE Transactions on Computer-Aided Design*, vol. 14, no. 11, pp. 1370-1378, Nov. 1995.

[50] I. Pomeranz, L. Reddy, and S. M. Reddy, "Compactest: A Method to Generate Compact Test Sets for Combinational Circuits," in *Proceedings of IEEE International Test Conference*, pp. 194-203, Oct. 1991.

[51] I. Hamzaoglu and J. H. Patel, "Test Set Compaction Algorithms for Combinational Circuits," in *Proceedings of IEEE International Conference on Computer-Aided Design*, pp. 283-289, Nov. 1998.

[52] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy, "Cost-Effective Generation of Minimal Test Sets for Stuck-at Faults in Combinational Logic Circuits," *IEEE Transactions on Computer-Aided Design*, vol. 14, no. 12, pp. 1496-1504, Dec. 1995.

[53] J. Saxena and D. K. Pradhan, "A Method to Derive Compact Test Sets for Path Delay Faults in Combinational Circuits," in *Proceedings of IEEE International Test Conference*, pp. 724-733, Oct. 1993.

[54] S. Kajihara, M. Fukunaga, X. Wen, T. Maeda, S. Hamada, and Y. Sato, "Path Delay Test Compaction with Process Variation Tolerance," in *Proceedings of ACM/ IEEE Design Automation Conference*, pp. 845-850, Jun. 2005.

[55] M. Fukunaga, S. Kajihara, X. Wen, T. Maeda, S. Hamada, and Y. Sato, "A Dynamic Test Compaction Procedure for High-Quality Path Delay Testing," in *Proceedings of IEEE Asia South Pacific Design Automation Conference*, pp. 348-353, Jan. 2006.

[56] C. J. Lin and S. M. Reddy, "On Delay Fault Testing in Logic Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, no. 5, pp. 694-703, Sept. 1987.

[57] K. -T. Cheng and H. C. Chen, "Classification and Identification of Nonrobust Untestable Path Delay Faults," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 8, pp. 845-853, Aug. 1996.

[58] S. Remersaro, J. Rajski, S. M. Reddy, and I. Pomeranz, "A Scalable Method for the Generation of Small Test Sets," in *Proceeding of Design, Automation & Test in Europe Conference & Exhibition*, pp. 1136-1141. Apr. 2009.

[59] J. P. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," *IBM Journal of Research and Development*, vol. 10, no. 4, pp. 278-291, Jul. 1966.

[60] L. H. Goldstein and E. L. Thigpen, "SCOAP: Sandia Controllability/Observability Analysis Program," in *Proceedings of ACM/IEEE Design Automation Conference*, pp. 190–196, Jun. 1980.

[61] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*. Springer, New York, 2000.

[62] Z. Wang and D. M. H. Walker, "Dynamic Compaction for High Quality Delay Test", in *Proceedings of IEEE VLSI Test Symposium*, pp. 243-248, Apr. 2008.

[63] M. H. Schulz and E. Auth, "Improved Deterministic Test Pattern Generation with Applications to Redundancy Identification," *IEEE Transactions on Computer-Aided Design*, pp.811-816, Jul. 1989.

[64] W. Kunz and D. K. Pradhan, "Accelerated Dynamic Learning for TEST Pattern Generation in Combinational Circuits," *IEEE Transactions on Computer-Aided Design*, vol. 12, no. 5, pp. 684-694, May 1993.

[65] W. Kunz and D. K. Pradhan, "Recursive Learning: A New Implication Technique for Efficient solution to CAD Problems - Test, Verification, and Optimization," *IEEE Transaction on Computer-Aided Design*, vol. 13, no. 9, pp. 1143-1158, Sept. 1994.

[66] J. Kibarian and A. Strojwas, "Using Spatial Information to Analyze Correlations between Test Structure Data," *IEEE Transactions on Semiconductor Manufacturing*, vol. 4, pp. 219–225, Aug. 1991.

[67] R. -S. Guo and E. Sachs, "Modeling, Optimization, and Control of Spatial Uniformity in Manufacturing Processes," *IEEE Transactions on Semiconductor Manufacturing*, vol. 6, pp. 41–57, Feb. 1991.

[68] B. Stine, D. Boning, and J. Chung, "Analysis and Decomposition of Spatial Variation in Integrated Circuit Processes and Devices", *IEEE Transactions on Semiconductor Manufacturing*, vol. 10, no. 1, pp. 24-41, Feb. 1997.

[69] S. Tani, M. Teramoto, T. Fukazawa, and K. Matsuhiro, "Efficient Path Selection for Delay Testing Based on Partial Path Evaluation," in *Proceedings of IEEE VLSI Test Symposium*, pp. 188-193, Apr. 1998.

[70] G. M. Luong and D. M. H. Walker, "Test Generation for Global Delay Faults," in *Proceedings of IEEE International Test Conference*, pp. 433-442, Oct. 1996.

[71] X. Lu, Z. Li, W. Qiu, W. Shi, and D. M. H. Walker, "Longest Path Selection for Delay Test Under Process Variation," in *Proceedings of IEEE Asian and South Pacific Design Automation Conference*, pp. 98-103, Jan. 2004.

[72] N. Ahmed, M. Tehranipoor, and V. Jayaram, "Timing-Based Delay Test for Screening Small Delay Defects", in *Proceedings of ACM/IEEE Design Automation Conference*, pp. 320-325, Sept. 2006.

[73] M. Amodeo and B. Cory, "Defining Faster-than-at-Speed Delay Tests," in *Nanometer Test Article*, Cadence Inc., Apr. 2005.

[74] W. W. Mao and M. D. Ciletti, "A Variable Observation Time Method for Testing Delay Faults," in *Proceedings of ACM/IEEE Design Automation Conference*, pp. 728-731, June 1990.

[75] B. Kruseman, A. K. Majhi, G. Gronthoud, and S. Eichenberger, "On Hazard-Free Patterns for Fine-Delay Fault Testing," in *Proceedings of IEEE International Test Conference*, pp. 213-222, Oct. 2004.

[76] B. Lee, L. -C. Wang, and M. Abadir, "Reducing Pattern Delay Variations for Screening Frequency Dependent Defects," in *Proceedings of IEEE VLSI Test Symposium*, pp. 153-160, May 2005.

[77] K. -T. Cheng and H. C. Chen, "Delay Testing for Non-Robust Untestable Circuits," in *Proceedings of IEEE International Test Conference*, pp. 954-961, Oct. 1993.

[78] W. K. C. Lam, A. Saldanha, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Delay Fault Coverage and Performance Trade-Offs," in *Proceedings of ACM/IEEE Design Automation Conference*, pp. 446-452, Jun. 1993.

[79] Z. Li, X. Lu, W. Qiu, W. Shi, and D. M. H. Walker, "A Circuit Level Fault Model for Resistive Bridges," *ACM Transactions on Design Automation of Electronic Systems*, vol. 8, no. 4, pp. 546-559, Oct. 2003.

[80] X. Lu, Z. Li, W. Qiu, D. M. H. Walker, and W. Shi, "A Circuit Level Fault Model for Resistive Shorts of MOS Gate Oxide," in *Proceedings of IEEE International Workshop on Microprocessor Test and Verification*, pp. 97-102, Sept. 2004.

[81] R. R. Montanes and J. P. Gyvez, "Resistance Characterization for Weak Open Defects," *IEEE Design & Test of Computers*, vol. 19, no. 5, pp. 18-26. Sept. 2002.

[82] J. B. Brockman and S. W. Director, "Predictive Subset Testing: Optimizing IC Parametric Performance Testing for Quality, Cost and Yield," *IEEE Transactions on Semiconductor Manufacturing*, vol. 2, no. 3, pp. 104–113, Aug. 1989.

[83] P. S. Zuchowski, P. A. Habitz, J. D. Hayes, and J. H. Oppold, "Process and Environmental Variation Impacts on ASIC Timing," in *Proceedings of IEEE International Conference on Computer-Aided Design*, pp. 336–342, Nov. 2004.

[84] Z. Li, X. Lu, W. Qiu, W. Shi, and D. M. H. Walker, "A Circuit Level Fault Model for Resistive Opens and Bridges," in *Proceedings of IEEE VLSI Test Symposium*, pp. 379-384, Apr. 2003.

[85] M. Spica, M. Tripp, and R. Roeder, "A New Understanding of Bridge Defect Resistances and Process Interactions from Correlating Inductive Fault Analysis Predictions to Empirical Test Results," in *Proceedings of IEEE International Workshop on Defect Based Testing*, pp. 11-16, Apr. 2001.

[86] Z. Wang and D. M. H. Walker, "Compact Delay Test Generation with a Realistic Low Cost Fault Coverage Metric," in *Proceedings of IEEE VLSI Test Symposium*, pp. 59-64, May 2009.

[87] M. Gowan, L. Biro, and D. Jackson, "Power Considerations in the Design of the Alpha 21264 Microprocessor," in *Proceedings of ACM/IEEE Design Automation Conference*, pp. 726-731, Jun. 1998.

[88] J. Wang, Z. Yue, X. Lu, W. Qiu, W. Shi, and D. M. H. Walker, "A Vector-Based Approach for Power Supply Noise Analysis in Test Compaction," in *Proceedings of IEEE International Test Conference*, pp. 517-526, Nov. 2005.

[89] J. Wang, X. Lu, W. Qiu, Z. Yue, S. Fancler, W. Shi, and D. M. H. Walker, "Static Compaction of Delay Tests Considering Power Supply Noise," in *Proceedings of IEEE VLSI Test Symposium*, pp. 235-240, May 2005.

[90] E. M. Rudnick and J. H. Patel, "Efficient Techniques for Dynamic Test Sequence Compaction," *IEEE Transactions on Computers*, vol. 48, no. 3, pp. 323-330, Mar. 1999.

[91] S. Y. Lee, B. Cobb, J. Dworak, M. R. Grimaila, and M. R. Mercer, "A New ATPG Algorithm to Limit Test Set Size and Achieve Multiple Detections of All Faults," in *Proceedings of IEEE Design, Automation and Test in Europe Conference and Exhibition*, pp. 94-99, Mar. 2002.

[92] J. Wingfield, J. Dworak, and M. R. Mercer, "Function-Based Dynamic Compaction and Its Impact on Test Set Sizes," in *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 167-174, Nov. 2003.

[93] N. A. Touba, "Survey of Test Vector Compression Techniques," *IEEE Design & Test of Computers*, vol. 23, no. 4, pp. 294-303, Apr. 2006.

[94] B. Koenemann, "LSFR-coded Test Patterns for Scan Designs," in *Proceedings of IEEE European Test Conference*, pp. 237-242, Apr. 1991.

[95] B. Koenemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater, "A SmartBIST Variant with Guaranteed Encoding," in *Proceedings of IEEE Asian Test Symposium*, pp. 325-330, Nov. 2001.

[96] J. Rajski, J. Tyszer, G.Mruglaski, W. -T. Cheng, N. Mukherjee, and M. Kassab, "X-Press Compactor for 1000x Reduction of Test Data," in *Proceedings of IEEE International Test Conference*, pp.1-10, Oct. 2006.

[97] F. F. Hsu, K. M. Bulter, and J. H. Patel, "A Case Study on the Implementation of Illinois Scan Architecture," in *Proceedings of IEEE International Test Conference*, pp.538-547, Oct. 2001.

[98] Z. Zhang, S. M. Reddy, and I. Pomeranz, "On Generating Pseudo-Functional Delay Fault Tests for Scan Designs," in *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 398-405, Oct. 2005.

[99] Y. -C. Lin, F. Lu and K. -T. Cheng, "Pseudofunctional Testing," *IEEE Transactions on Computer Aided-Design of Integrated Circuits and Systems*, vol. 25, no.8, pp. 1535-1546, Aug. 2006.

[100] M. Syal, K. Chandrasekar, V. Vimjam, M. S. Hsiao, Y. -S. Chang, and S. Chakravarty, "A Study of Implication Based Pseudo Functional Testing," in *Proceedings of IEEE International Test Conference*, pp. 1-10, Oct. 2006.

[101] W. Wu and M. S. Hsiao, "Mining Sequential Constraints for Pseudo-Functional Testing," in *Proceedings of IEEE Asian Test Symposium*, pp. 19-24, Oct. 2007.

[102] W. Qiu, X. Lu, Z. Li, D. M. H. Walker, and W. Shi, "CodSim - A Combined Delay Fault Simulator," in *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp. 79-86, Nov. 2003.

[103] K. L. Shepard and V. Narayanan, "Noise in Deep Submicron Digital Design," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pp. 524-531, Nov. 1996.

[104] Z. Jiang, Z. Wang, J. Wang, and D. M. H. Walker, "Realistic Low Cost Framework for Supply Noise-Aware Delay Test Compaction," in *IEEE Workshop on Defect and Data Driven Testing*, sec. 3.3, Nov. 2009.

VITA

Zheng Wang

Dept. of Computer Science and Engineering, Texas A&M University,

College Station, TX, 77843-3112

United States of America

E-mail: philwz79@hotmail.com

Zheng Wang was born in Luzhou, China. He obtained his B.S. in information science & electronic engineering and M.S. in communication and information system from Zhejiang University, Hangzhou, China in July 2001 and April 2004 respectively, and a Ph.D. in Computer Engineering from Texas A&M University, College Station, TX in May 2010. He worked as an integrated circuit (IC) design engineer for Philips Semiconductor (currently NXP) Shanghai from May 2004 to July 2006 before joining Texas A&M University for his doctoral studies in August 2006. His research interests are delay fault testing, automatic test pattern generation, design-for-test and test compaction. His non-technical interests include music, badminton, tennis, travel and photography. He is a member of IEEE.